NAS 7820 and NAS 7821 Software Overview

Version 1.00

June 22 2010

SG-0027

Website	www.plxtech.com
Technical Support	www.plxtech.com/support

Document number: SG-0027

Contents

This page is intentionally blank

Preface

This guide gives architectural information about the NAS 7820 (single SATA port, order code NAS 7820-AABC F) and NAS 7821 (dual SATA port, order code NAS 7821-AABC F) devices. The devices are jointly referred to as NAS 782[0/1]. It also gives a high-level overview of the software for a NAS system.

This guide assumes that readers are software engineers who are experienced in developing software applications using C/C++, are familiar with Linux operating systems and have an understanding of Ethernet and networking issues.

Typographic Conventions

In this document, the following conventions apply.

Convention	Meaning
Italic Letters With Initial Capital Letters	A cross-reference to another publication
Title	A cross-reference to another section within the document
1, 2, 3	A numbered list where the order of list items is significant
■	A list where the order of items is not significant
⚡	Significant additional information
`Courier`	Software code
Bold	Significant names, for example of files or directories Text you type

Revision Information

The following table documents the revisions of this guide.

Version	Date	Modification
1.00	June 22 2010	New document

This page is intentionally blank

Term	Meaning
AHB	AMBA High-performance Bus, where AMBA is Advanced Microcontroller Bus Architecture
ARM	Advanced RISC Machines
BOOTP	Bootstrapping protocol
buildroot	Used to build a root file system and user-space applications to run on the NAS device
CIFS	Common Internet File System
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CUPS	Common Unix Printing System. A printing system allowing a computer to act as a print server
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol. A protocol for allocating IP addresses dynamically to computers on a local area network. The system administrator assigns a range of IP addresses to DHCP and each client computer on the LAN has its TCP/IP software configured to request an IP address from the DHCP server. The request and grant process uses a lease concept with a controllable time period
DMA	Direct Memory Access
DNS	Domain Name System
DRAM	Dynamic Random Access Memory
EXT3	Third extended file system: a journaled file system used by the Linux kernel
FAT32	File Access Table, 32 bit: a Microsoft file system
GMAC	Gigabit MAC
GPIO	General Purpose I/O
HFS+	Hierarchical File System Plus: an Apple Macintosh file system
HTML	HyperText Markup Language
I^2C	Inter-Integrated Circuit: a multi-master serial single-ended computer bus
IDE	Integrated Drive Electronics. A computer hardware bus used primarily for hard drives and optical drives
IEEE 802.11	Wi-Fi protocol (various subdivisions for speed and throughput)
I/O	Input/output
IP	Internet Protocol

Term	Meaning
ISO	International Standards Organizaiton
Linux kernel	Linux operating system used for the root file system in the NAS application
MAC	Media Access Control
MBR	Master Boot Record
MMU	Managed Memory Unit
NAS	Network-Attached Storage
NTFS	New Technology File System, a Microsoft file system
NTP	Network Time Protocol
PCIe	Peripheral Component Interconnect Express
PLL	Phase-Locked Loop
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RARP	Reverse Address Resolution Protocol
ROM	Read-Only Memory
RPS	Reference Peripheral Set
SATA	Serial Advanced Technology Attachment
Samba	A networking protocol providing file sharing and print server facilities in the NAS application
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Static Random Access Memory
Stage 1 Loader	U-Boot loader when booting from hard disk drive
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UART	Universal Asynchronous Receiver/Transmitter
U-Boot	Boot loader and utility to create loadable Linux images
UDP	User Datagram Protocol
URL	Uniform Resource Locater: a web page address
USB	Universal Serial Bus
XFS	A journaling file system, originally developed by SGI

Introduction

The NAS 782[0/1] network-attached storage (NAS) device offers a range of features, including RAID, disk-spanning and advanced encryption capabilities, to take information from a network and preserve it securely on a local disk. The NAS 782[0/1] offers diverse connectivity including 10/100/1000 Ethernet plus CIFS over TCP/IP. The NAS 782[0/1] also supports connections to IEEE 802.11 compliant networks and other standard wireless chips using PCIe, with integrated SATA cores (expandable using a PCIe bus), two USB 2.0 host ports, and expansion capability using USB-attached disks. The integrated ARM processor, featuring an MMU and caches, runs Linux with a full network stack.

NAS devices are configured by pin multiplexing, which enables you to select functions from a range of possibilities, and are supported by firmware built specifically for the target system. This extends the capacity of the devices for off-chip control, for example fan control or facilitating low-resolution output monitoring.

The NAS 782[0/1] also offers facilities for imposing password protection on a system using a web interface. The web interface allows you to configure and manage the system, and enables you to create and maintain further web pages for customizing system features.

Development Kit Elements

The ISO image contains the following software elements and support functions:

- Linux kernel
- Bootloader
- Root file system
- Web Interface
- System Components

These help you to manage and coordinate a complete network-attached-storage product.

Web Interface

The web interface includes the following elements:

- Lighttpd web server (small, fast and with built-in authentication)
- Web pages you can configure, with Perl scripts to implement changes to the system:
    - Initial pages—with end-user license agreement and initial configuration items
    - Home page—links to configuration and sharing pages
    - Configuration page—allows system configuration of:
        - Date and time, including time zone selection
        - Network, including automatic or static IP selection and workgroup/netmask
        - Network device name
        - Language (the web pages support multiple languages, although other language strings are not shipped as part of the development kit)
        - Disk format/RAID rebuild
        - USB safe removal
    - Sharing page—sharing functions such as:
        - User configuration: add users, delete users, rename users
        - Create shares, such as Samba shares
        - Rename shares
        - Delete shares
        - Change share access rights

System Components

The NAS system components are a group of utilities governing fundamental device behavior and usability:

- Software architecture
    - Linux
    - TCP/IP and CIFS
    - uClibc
    - EXT3 or XFS file system
    - Boot from hard disk drive (no external flash memory needed), or boot from flash
- Simple installation
    - DHCP client
    - NTP client
    - Support for real-time clock device using I^2C interface

- Single or dual hard disk drive systems
    - Software stored on hard disk drive
    - Dual hard disk drive system has redundancy of system software, allowing the system to continue to operate following a single hard disk drive failure
    - Hard disk drive format and automatic RAID rebuild function
- USB expansion
    - Support for additional storage using USB drive
    - Support for flash cards
    - Safe removal function
- Upgrade function
    - Web page and Perl scripts available for firmware upgrade
    - Mechanism needs a server to be hosted by customer
    - Server must decide whether new firmware is available based on version number sent by NAS system

Configuration Supplied on the ISO Image

As supplied, the development kit has the following features:

- The system boots from disk; the disk must be attached to SATA port A
- The system is configured as a single hard disk drive NAS system. The system can also make use of a USB 2.0 port to attach an additional hard disk drive to expand storage capacity
- The IP address is set automatically, either using a DHCP server or, if a server is unavailable, using zeroconf.

  Device name is **OXNAS**.
- The UART can be used for application debugging with a PC application such as **cu**

  User name: root
  Password: root
- Web page administrator details:

  User name: admin
  Password: 123456

This page is intentionally blank

NAS Software

This chapter explains more about the NAS system software. It covers the following topics:

- Development Kit Software Elements—see below
- Software Overview—see page 6

Development Kit Software Elements

Table 2-1 details the NAS software element hierarchy in the development kit.

Table 2-1 NAS Software Constituents

Element	Description
Buildroot	Used to build a root file system and user-space applications to run on the NAS device
Linux kernel	Version 2.6.31. Used for the root file system in the NAS application
Samba	Provides file sharing and print server facilities in the NAS application
U-Boot	Boot loader and utility to create loadable Linux images
Stage 1 Loader	U-Boot loader when booting from hard disk drive

Software Overview

A NAS system comprises the following elements:

- Stage 1 loader
- U-Boot
- Linux kernel
- Root file system—contains the operating system, user applications and web user interface

ROM Loader

The ROM loader can boot from either SATA disk or flash memory. In boot-from-SATA mode, the ROM loader accesses locations in the master boot record (MBR) of the disk that are normally unused. The locations identify sectors containing U-Boot, which must be loaded from disk into internal SRAM and executed.

Stage 1 Loader

The Stage 1 loader reads U-Boot when booting from a SATA disk and configures the DDR memory.

The DDR core supports a range of SDRAM memory devices and sizes. Code that configures the DDR controller cannot itself use the SDRAM, because changes to the DDR controller configuration cause invalid data to be read from or written to the SDRAM, which in turn causes invalid code execution. The ROM loader does not include DDR controller setup code, because the memory devices that require support cannot be predicted in advance; for this reason a later stage of system startup must perform the memory configuration.

In boot-from-SATA mode, Stage 1 Loader performs DDR controller configuration while executing from SRAM, and then loads U-Boot into the SDRAM.

For detailed register definitions of the DDR controller, see the programming guide for the device.

In the development kit, the DDR memory configuration is defined in the software and the correct memory size is passed to Linux using its boot arguments.

U-Boot

U-Boot performs the following tasks:

- Clock enabling and configuration
- Reads the kernel from the hard disk drive

Clock Control

U-Boot enables and configures a number of internal clock domains. PLLs are used to create all internal clocks, as follows:

- PLL A—multiplies the 25MHz crystal input to (typically) 750MHz
- PLL B—multiplies the 25MHz clock to produce the 60MHz required for the USB core

Table 2-2 lists the clocks used by NAS 782[0/1] devices, their derivation and default speeds.

Table 2-2 NAS 782[0/1] Clocks, Derivation and Default Properties

Clock Name	Default Speed	Enabled/Disabled in the Boot Loader
CPU clock	750MHz	Enabled
DDR Clock	375MHz	Enabled
DMA	187MHz	Disabled
SATA system clock	187MHz	Enabled
GMAC core	187MHz	Disabled
Network coprocessor (Leon)	375MHz	Disabled
Static core	187MHz	Enabled
Cipher	187MHz	Disabled
USB AHB	187MHz	Disabled
USB core	187MHz	Disabled
RPS clocks	6.25MHz	Enabled

DDR Controller Configuration

The boot sequence can be halted while still in U-Boot by pressing any key using the debug serial interface. Table 2-3 shows which cores are enabled at this point.

Table 2-3 Cores Enabled Following U-Boot

Core	Notes
CPU	I-cache is enabled D-cache is disabled MMU is enabled
DDR	U-Boot code runs out of the DDR memory
DMA	Includes scatter-gather functionality
SATA	Runs at 1.5G by default
Internal SRAM	All 64Kbytes is available
USB core	Disabled
PCIe core	Enabled, but only for static bus arbitration
Network coprocessor	Disabled
GMAC	Enabled; tftp is available from U-Boot
UART	Enabled; debug information is output from UART 2
GPIO	Subset of the available GPIOs is enabled

A number of commands are built into U-Boot, and they can be run from the U-Boot command prompt displayed following an interruption in the U-Boot process. The commands supported by default are listed in Table 2-4.

Table 2-4 Supported U-Boot Commands (Sheet 1 of 2)

Command	Action
base	Print or set address offset
bdinfo	Print board information structure
bootm	Boot application image from memory
bootp	Boot image from network using BOOTP/TFTP protocol
cmp	Memory compare
cp	Memory copy
crc32	Checksum calculation
diskboot	Boot from IDE device
echo	Echo arguments to console
exit	Exit script

Table 2-4 Supported U-Boot Commands (Sheet 2 of 2)

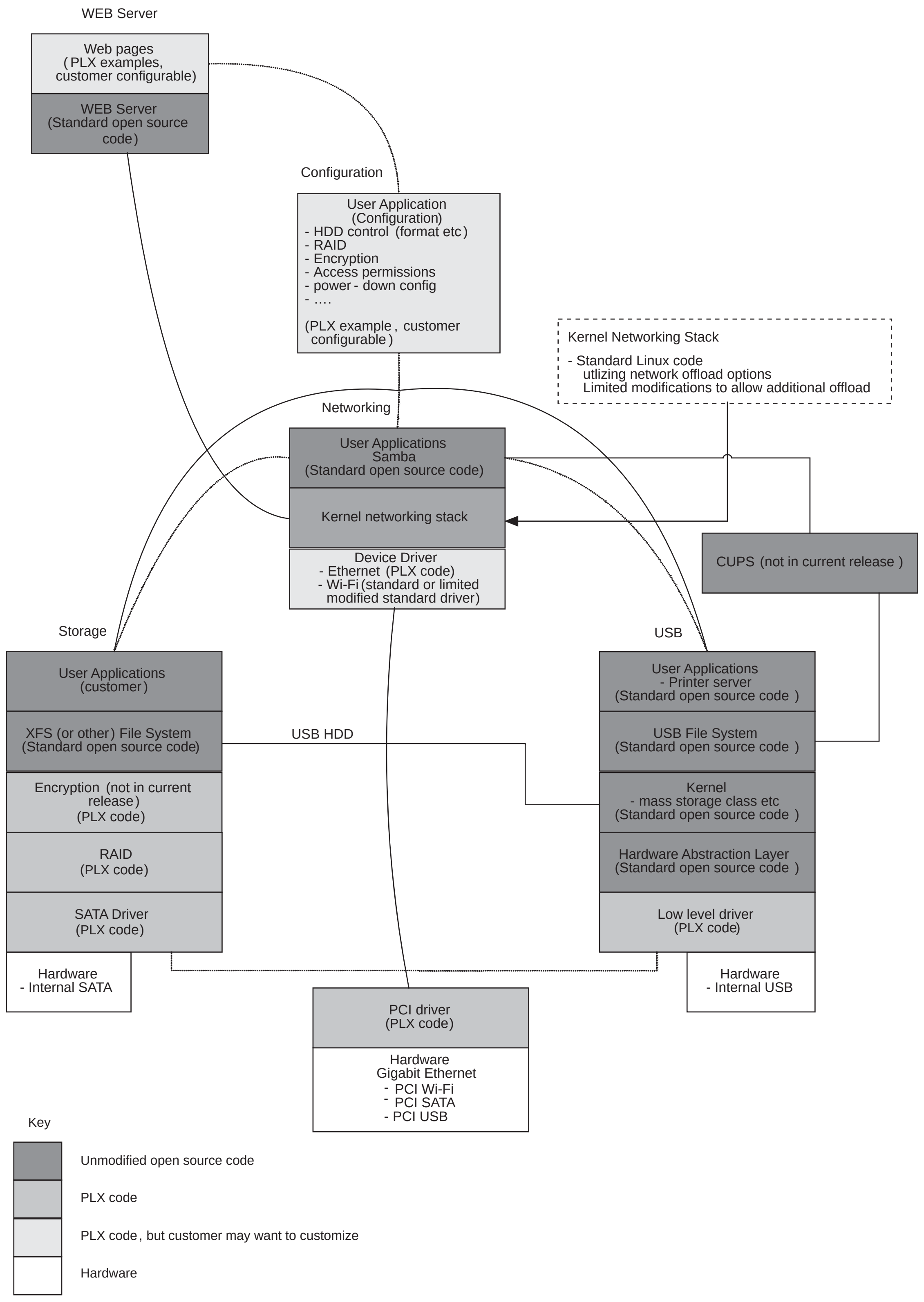
Command	Action
ext2load	Load binary file from an EXT2 file system
ext2ls	List files in a directory (default /)
go	Start application at address 'addr'
help	Print online help
ide	IDE subsystem
iminfo	Print header information for application image
loop	Infinite loop on address range
md	Memory display
mm	Memory modify (auto-incrementing)
mtest	Simple RAM test
mw	Memory write (fill)
nm	Memory modify (constant address)
printenv	Print environment variables
rarpboot	Boot image over network using RARP/TFTP protocol
reset	Perform CPU reset
run	Run commands in an environment variable
saveenv	Save environment variables to persistent storage
setenv	Set environment variables
test	Minimal test like /bin/sh
tftpboot	Boot image over network using TFTP protocol
version	Print monitor version

Linux Kernel

Figure 2-1 on page 10 illustrates the software modules that comprise the Linux kernel and user application system and shows how they interact.

For NAS 7820 and NAS 7821 devices with dual ARM11 processors, Symmetric MultiProcessing (SMP) Linux is supplied.

Figure 2-1 Linux Kernel and User Application Modules


WEB Server
Web pages (PLX examples, customer configurable)
WEB Server (Standard open source code)
Configuration
User Application (Configuration) - HDD control (format etc) - RAID - Encryption - Access permissions - power - down config - ….
(PLX example, customer configurable)
Kernel Networking Stack
- Standard Linux code utlizing network offload options Limited modifications to allow additional offload
Networking
User Applications Samba (Standard open source code)
Kernel networking stack
Device Driver - Ethernet (PLX code) - Wi-Fi (standard or limited modified standard driver)
CUPS (not in current release)
Storage
User Applications (customer)
XFS (or other) File System (Standard open source code)
USB HDD
Encryption (not in current release) (PLX code)
RAID (PLX code)
SATA Driver (PLX code)
Hardware - Internal SATA
USB
User Applications - Printer server (Standard open source code)
USB File System (Standard open source code)
Kernel - mass storage class etc (Standard open source code)
Hardware Abstraction Layer (Standard open source code)
Low level driver (PLX code)
Hardware - Internal USB
PCI driver (PLX code)
Hardware Gigabit Ethernet - PCI Wi-Fi - PCI SATA - PCI USB
Key
Unmodified open source code
PLX code
PLX code, but customer may want to customize
Hardware

The NAS system is based on Linux 2.6.17.14.

The kernel is the central component of the operating system. Its responsibilities include managing system resources and handling communication between hardware and software components. As a basic component of an operating system, a kernel provides the lowest level abstraction layer for the resources (especially memory, processors and I/O devices) that applications must control to perform their function. It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls.

The kernel and hardware drivers provide the mechanisms for the following tasks:

- Communicating with other systems (using the Ethernet driver, running on the internal GMAC, using the network stack)
- Controlling a USB storage device (using a standard USB driver and mass storage class USB stack)
- Providing system resources and tools to higher-level applications

The kernel in the NAS 782[0/1] is built on layers of software with increasing abstraction away from physical devices:

1. The top of the abstraction is the interface to user space application using sockets, virtual subsystems which control the virtual memory system and process execution
2. Internal subsystems including network protocol stack (including TCP, UDP and IP), logical files systems and task management
3. The hardware abstraction includes the block device, memory manager and operating system scheduler
4. Hardware drivers including bus drivers, network drivers, SATA driver and interrupts

User space applications interface to various functions:

- Process scheduler which controls process execution, interfacing to CPU registers
- System device driver that includes a number of bus drivers including USB and PCIe
- Network stack which interfaces between Ethernet devices and higher-level protocol stacks
- Storage stack that abstracts from the low-level SATA driver to the virtual file system
- Memory system which uses RAM and MMU at the lowest level, abstracting to the virtual memory manager

Root File System and User Applications

The root file system holds the configuration data and executable files needed for higher level applications. It supports a number of user applications including Samba, the web user interface and terminal access software for debugging.

Samba

Samba provides an open-source common internet file system (CIFS), which allows the NAS to expose the device server name and provide file shares so that PCs on the network can transparently use the files.

Web User Interface

The NAS system contains a web server, called **lighttpd**, which supports the password authentication process for logging in.

The NAS web interface provides a convenient way of configuring and managing the device, using a standards-compliant web browser. The web interface configuration application, which is used to set up the interface, is multilingual and the page layout can easily be localized.

The web pages are based on templates, with borders created from standard templates and a central part for each page. You can configure the web pages for your system.

Implementing a NAS System

This chapter covers the following major topics:

- Updating the root file system—see page 13
- Creating a bootable disk—see page 14
- Upgrading the firmware—see page 14

Updating the Root File System

For instructions on replacing the root file system, see the relevant getting started guide for your device.

Booting a NAS System

The NAS process for booting from a SATA disk is as follows:

1. On power-on-reset, the processor starts running instructions from the internal boot ROM. The processor reads specific boot-mode pins on the chip, because the ROM has a number of boot modes. It configures the SATA core and reads the Stage 1 Loader from specific locations on the hard disk drive.

2. The boot ROM runs a checksum on the Stage 1 Loader. If the image fails its checks, the ROM loader fails over to a secondary Stage 1 Loader image on the disk. If the secondary image is also corrupt, the other SATA drive is tried on dual-disk systems. If the checks are successful, the Stage 1 Loader is loaded into internal SRAM and executed. Its job is to read U-Boot from the disk.

3. The Stage 1 Loader runs a checksum on U-Boot. If the image fails its checks, the Stage 1 Loader fails over to a secondary U-Boot image on the disk. If the secondary image is also corrupt, the other SATA drive is tried on dual-disk systems. If the checks are successful, U-Boot is loaded into internal SRAM and executed.

   Environment data for U-Boot is also loaded into SRAM at this point, again from redundant areas on each disk.

4. U-Boot initializes its environment data from what was loaded into SRAM, copies a Linux kernel image from the disk into DRAM memory, then unpacks and starts the kernel. U-Boot tasks are explained in more detail in U-Boot on page 6.

As the system boots, displays indicate the stages of the booting process.

In boot-from-flash mode, the boot ROM loads U-Boot from flash memory, which in turn loads the Linux kernel from flash, unpacks it and boots the kernel.

More information about U-Boot is available from http://www.denx.de/wiki/DULG.

Creating a Bootable Disk

For guidance on how to create the disk image for the NAS, see the relevant getting started document for your device.

Hard Disk Drive System Partitioning

The system software is stored in multiple partitions on the hard disk drive.

Hidden Sectors Layout

Table 3-1 summarizes how the hidden sectors on the disk are used.

Table 3-1 Hidden Sectors

Sector	Contents
0	MBR
34	Stage 1 loader
154	U-Boot
1290	uImage (Linux kernel)
8482	Upgrade uImage
16674	Upgrade root file system
57088	Secondary Stage 1 loader
57208	Secondary U-Boot
58344	Secondary uImage

Upgrading the Firmware

The firmware upgrade function automatically checks whether new firmware is available from your upgrade server (hosted by you at a specified web address). The NAS system sends the version number to the server, which must decide whether new firmware is available; if so, it provides the location of the upgrade script.

The upgrade process is controlled from the user interface. The user interface downloads an archive containing the revised system and starts the upgrade. The current archive is contained in a self-extracting format and it includes a list of the MDS check values for the upgrade file. The upgrade is aborted if the calculated MDS digest values do not agree with the list values.

A statically linked environment for the RAM disk hosts part of the upgrade process so that the root file system partition can be reformatted. The initial RAM disk image and a reduced function kernel is stored in a number of hidden disk sectors; U-Boot uses these images to boot when upgrade mode is selected.

The upgrade process is controlled using flags stored at various times during the process. This ensures that the system can maintain a good state even if power is lost during the upgrade.

Details of the upgrade process are given in Chapter 4 Upgrading Software in the Field.

This page is intentionally blank

Upgrading Software in the Field

This chapter explains the NAS system upgrade process, with particular reference to the dual-disk NAS system.

The facilities to do this are NOT included in the early kernel release.

Upgrade Process Overview

There are several stages to the upgrade process, which is outlined below:

1 You use a web browser to select the upgrade page on the NAS administration pages.

2 The NAS system contacts the upgrade server using the internet and provides a search string containing the product code and current firmware issue state.

3 The upgrade server provides the NAS with a URL of the required upgrade archive file, or the message “No upgrade available” in the HTML body when the NAS is running the latest firmware.

4 The NAS continues to download the identified file into a defined location on the NAS system in the private data area. It is not available in any accessible shared directory.

5 The upgrade process unpacks the first level of the archive, making available a script which, when run, coordinates the update process.

Other components of the archive may include an updated U-Boot loader, kernel and/or root file system. If loader components are present, they are copied to the positions in the hidden cylinders at the start of the disk reserved for them and are used on the next boot.

6 If a new root file system was provided, the upgrade script causes the NAS to reboot into the initial RAM disk that has been written to a location in the hidden cylinders. This allows the NAS to boot without using the disk-based root file system. In addition, it allows the partition holding the root file system to be repartitioned and formatted with the contents of the new root file system image. When the new contents have been established, the system reboots into the upgraded disk-based file system and normal operation resumes.

Before You Start

The upgrade process operates on a reboot-into-upgrade-mode method. A statically linked environment for the RAM disk hosts part of the upgrade process so that the root file system partition can be reformatted. The initial RAM disk image and a reduced function kernel is stored in a number of hidden disk sectors. U-Boot uses them to boot when upgrade mode is selected.

The update process is controlled from the user interface. The user interface downloads an archive containing the revised system and starts the upgrade. The current archive is contained in a self-extracting format. After unpacking, the script **upgrade1-xdelta.sh** is executed to allow a binary difference file to update the root file system, reducing the size of the upgrade file; the installed versions are retained in the **/var/lib** directory. When the difference file update has completed, the md5sum values are calculated and compared with the values listed as part of the upgrade file. If they do not agree, the upgrade is aborted. After md5 value verification, the script **update1.sh** is executed. It copies the files in the upgrade package to their working locations, rebooting into upgrade mode to change the kernel and root file system images. When upgrade mode is complete, the system is rebooted and resumes operating as a working system. As part of the reboot startup initialization, the script **update2.sh** executes and is then deleted.

System Partitioning

The two internal SATA drives have four partitions as described in Hard Disk Drive System Partitioning on page 14.

Functionality of the Loader Components

There are the following major loader components:

- ROM Loader
- Stage 1 Loader
- U-Boot

ROM Loader

The ROM loader decides from which disk and from which of the redundant images on that disk to load the Stage 1 loader, based on the integrity of the image. The Stage 1 loader image is prefixed with an image length and CRC header to provide for integrity detection.

The ROM loader initially loads the image and then verifies image correctness in memory before allowing execution. The ROM loader fails over from the first disk primary image to the secondary image, then to the second disk. There is no recovery if a valid image cannot be found, but the loader retries both disks until power off.

Stage 1 Loader

The Stage 1 loader initially loads the U-Boot image and then verifies image correctness in memory before allowing execution. The U-Boot image is prefixed with an image length and CRC header to provide for integrity detection. The Stage 1 loader fails over from the first disk primary image to the secondary image, then to the second disk. There is no recovery if a valid image cannot be found, but the loader retries both disks until power off.

U-Boot

U-Boot checks the integrity of **uImage** (the Linux kernel image) in primary and secondary locations on each disk in a similar way to the ROM loader U-Boot image checks. U-Boot uses an upgrade flag to decide which system to load and execute. The upgrade mode system is contained in two U-image data files, one for the kernel and one for the initial RAMdisk. During normal operation the kernel image is loaded from a U-boot data file into memory and executed. The root file system is provided by the device **/dev/md1**.

Upgrade Process

The upgrade process succeeds even if a single disk drive fails during the upgrade process. The processes described in this section cover normal mode (see below) and upgrade mode (see page 21).

Normal Mode

If the system is not operating in RAID1 for the rootfs, swap and persistent partitions, you are prevented from initiating an upgrade. In addition, the U-Boot and uImage files on both disks must be valid.

The persistent partition contains a record of the most recent successful upgrade. Success is recorded in log **/var/log/upgrades** and the current version is stored in **/var/lib/current-version**.

Use the user interface to initiate a download of the most recent upgrade archive to the NAS—specifically to the upgrade directory in the persistent partition. The integrity of this archive is checked by unpacking the archive, reconstructing the upgrade files from the downloaded archive contents and the installed copies, then verifying the MD5 Digest values of the reconstructed files. The current archive format is a self-extracting archive that uses the Unix tools **tail**, **tar** and **gunzip** to decompress the archive into its constituent files. The command **xdelta3** is used to merge the difference files in the downloaded upgrade archive with copies of the installed image files. md5sum is used to validate the unpacked and reconstructed files against the downloaded list of MD5 digest values.

The process produces the following files for a full update:

File	Description
md5sum.lst	List of md5 sum values
upgrade1.sh	Script to be run to perform the first steps of the upgrade process
upgrade2.sh	Script to be run to perform the final steps of the upgrade process
initrd	An initial RAMdisk image containing the entire environment for the final stage of the upgrade process encapsulated as a U-Boot-loadable image
uImage.1	A kernel image compatible with the upgrade initial RAMdisk
stage1.bin	An optional new stage1 loader image
u-boot.bin	An optional new U-Boot image and optionally environment
uImage	An optional new kernel image
rootfs.ext2.gz	An optional new Linux root file system image
upgrade1-xdelta.sh	Script to be run before download file validation, to merge differential upgrade files with existing files stored on the NAS

upgrade1-xdelta.sh and **upgrade1.sh** are mandatory items, because the next step in upgrading is started by calling them. The script upgrade1-xdelta.sh is called to reconstruct the complete set of upgrade files. The upgrade files are then verified by the internal **md5sum** program before calling upgrade1.sh to continue.

Executing the supplied **upgrade1.sh** script causes the following actions:

- Upgrade initial RAMdisk image file **uUpgradeRootfs** to each disk in turn
- Upgrade kernel **uImage** file to each disk in turn
- When the root filing system is being upgraded, set the upgrade mode flag for U-Boot on both disks, detecting and dealing with failure. If a failure occurs at this stage preventing the upgrade flag from being set on the second disk, then the upgrade flag on the first disk is cleared to abort the upgrade

Upgrade Mode

The system is rebooted, causing U-Boot to enter upgrade mode and also the following actions:

- The new Linux kernel version is copied to the hidden sectors on each disk in turn
- The RAID volume holding the root file system is reformatted, using the RAID device **/dev/md1**. This process only fails if there is a two-disk failure or a power failure. A power failure causes the upgrade process to begin again from booting into upgrade mode. A single-disk failure does not stop the formatting operation, but the failed disk partition is ejected from its RAID set
- The new root file system is unpacked from **/var/upgrade/rootfs.tar.gz** to the newly formatted root file system RAID1 partition, **/dev/md1**
- The upgrade mode flag to U-Boot on both disks is cleared. Even if there is a power failure prior to both flags being cleared, the system boots up in upgrade mode on the next boot and executes the upgrade again
- The system reboots into normal operation
- The upgrade is recorded in the upgrade log and the new version is stored in **/var/lib/current-version**
- All upgrade-related files are deleted from **/var/upgrade**

Partition for Persistent System State

The RAID1 partition **/dev/md3** mounted as **/var** contains the persistent system data.

Upgrade Status and Progress

The **/var** directory hosted on the persistent RAID1 volume maintains information about upgrade state in the **/var/upgrade**, and **/var/lib** directories. The state consists of two components:

- Storage for the version of the most recent upgrade applied successfully, in the form of a file holding the upgrade major and minor version numbers as ASCII text in **/var/lib/current-version**
- Storage for information about any upgrade in progress is in **/var/upgrade**

When the system completes booting into normal operating mode and discovers a successfully completed upgrade, it updates the record of most-recently applied upgrade and deletes all information related to the just completed upgrade.

This page is intentionally blank

Default System Configurations

SATA

The device is configured to run as a single-SATA device and only boots from SATA port A.

File Systems

The file system used for the internal SATA drives is configured as part of the software release. The file system is Linux-supported, for example EXT3 or XFS.

USB devices can support other file formats:

- FAT32 or FAT16—old file system, not recommended, though it has the advantage that it can be read by a Windows-based PC. Both writes and reads are supported for FAT32 and FAT16 formatted devices
- NTFS—New Technology File System. A Windows file system. Linux can read NTFS file systems but not write when using the NTFS kernel module-based driver. To obtain write ability, use the NTFS-progs-based driver
- Apple HFS+—Apple file system with Linux support. Read/write is supported, but when journalling is enabled, support is read only
- EXT2 or EXT3— Linux file system. Both writes and reads are supported for EXT2 and EXT3 formatted devices

Network Address Discovery

This is used to advertise the NAS to other devices on a network.

- DHCP—default dynamic host configuration protocol—automatically gets an IP address from DHCP server
- Zero Config—provides a range of default addresses that the system attempts to use if the DHCP fails
- Static IP address—fixed address that is defined by the user for booting directly after the disks are programmed

IP Address Flow

On power-up, the software attempts to define the IP address using a number of mechanisms:

1. If a static address is defined, this address is always used.
2. When a valid connection is available, it uses DHCP to obtain an address from a DHCP server.
3. If a DHCP server is not available, it uses a zero conf address and tries to establish whether this address is in current use. When it finds an address that is not being used by another device, it assumes that address.

System Software

System software used includes:

- CIFS—from Microsoft; used to expose device server name and provide file and printer shares so that PCs on the network can transparently use the files and printers
- Samba—a CIFS server

Network Control Scripts and Other Services

Network Control Scripts

The development kit supports IP address acquisition using DHCP if a server is available. It can also allow the automatic assignment of a gateway address, time server address and DNS server address. If a DHCP server cannot be found on booting, the system reverts to the zero conf protocol to obtain a unique link local IP address. The development kit also supports an alternative networking mode where the IP address and gateway address and other variables are statically defined by configuration files.

The file **/var/oxsemi/network-settings** defines several networking-related parameters; its default contents are reproduced below:

```
system_type=2NC
network_mode=dhcp
hostname=OXNAS
workgroup=workgroup
static_ip=169.254.1.10
static_msk=16
static_gw=
static_dns1=
static_dns2=
static_dns3=
static_ntp=
revert_to_dhcp=no
```

If network_mode is set to dhcp, booting causes the system to attempt to reach a DHCP server; the system falls back to zero conf if no DHCP server can be found.

If network_mode is set to static, the IP address and mask, gateway address, timer server (ntp) address and up to three DNS server addresses are sourced from the appropriate entries in the same file.

A special parameter, revert_to_dhcp, is effective in static networking mode. If this is set to **yes**, on the next boot of the system the networking mode is changed to DHCP. This feature is invaluable in production, where systems are initially required to boot with a well-known static IP address to allow production tests to run, then on subsequent boots normal DHCP must be used.

This page is intentionally blank