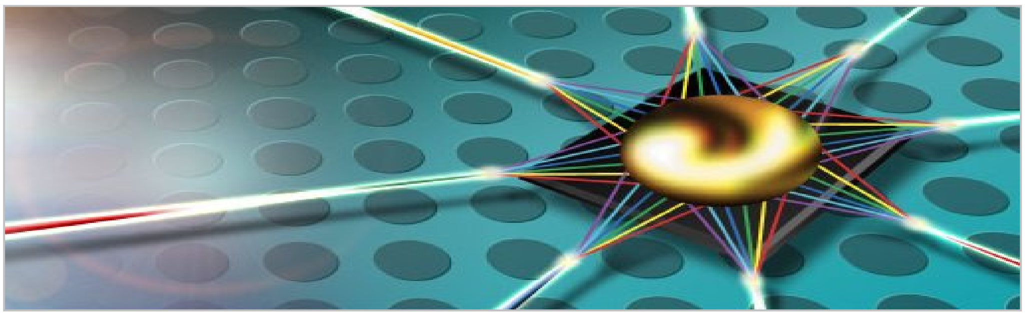
NPS-400

Software Load Sequence

Overview of the Process of Loading Software onto the NPS-400

Document Version 1.0



Document Number: 27-8249-00

EZchip Semiconductor Ltd.
1 Hatamar Street, P.O. Box 527
Yokneam 20692, Israel

EZchip Semiconductor Inc.
2700 Zanker Road, Suite 150
San Jose, CA 95134, USA

Preface

EZchip Semiconductor Inc.
2700 Zanker Road, Suite 150,
San Jose, CA 95134, USA
Tel: (408) 520-3700, Fax: (408) 520-3701

EZchip Semiconductor Ltd.
1 Hatamar Street, PO Box 527,
Yokneam 20692, Israel
Tel: +972-4-959-6666, Fax: +972-4-959-4166

Email: info@ezchip.com, Web: www.ezchip.com

Please send comments regarding the documentation and/or product to supportNP@ezchip.com.

About this Manual

This document provides a high level overview of the process of loading software onto the NPS-400.

This document includes by reference the "Power-up and Initialization" section of the *NPS-400 Architectural Specification* document and augments it with software and application oriented details.

This document geared towards software developers, application developers, and system architects. It assumes that they are familiar with several open source projects such as U-Boot, Linux, and Buildroot.

Related Documentation

For additional information refer to:

DOCUMENT	***CONTENT***
NPS-400 Architectural Specifications	Overview of the architecture, feature set and functionality of the NPS-400 network processor. In particular refer to the "Power-up and Initialization" section.
CPE Developer's Guide	Describes the EZdk Control Plane Environment (CPE) libraries used to development control plane applications for systems based on NPS-400 network processors.
NPS-400 Programming Manual	Offers programmers with the basic information required to code a data plane processing application for the EZchip NPS-400 network processor.
NPS-400 Hardware Specifications	Hardware specifications for the NPS-400 network processor, including pinout, electrical specifications, mechanical specifications and timing.
NPS-400 Hardware Design Guide	Provides hardware guidelines and recommendations for designing boards with the NPS-400 network processor.
EZtcp Stack Programming Manual	Offers programmers with the basic information required to use the TCP/IP stack package in a data plane processing application for the NPS-400.
NPS-400 EZdp Reference Manual	Describes the EZchip Data Plane Services library (EZdp) and its related APIs. The EZdp library provides an application programming interface (API) for data-plane applications running on NPS-400 network processors, abstracting the complexities of the underlying CTOP core instruction set and various hardware accelerators.

This Document

The following is a brief description of the contents of each chapter:

CHAPTER	***NAME***	***DESCRIPTION***
Section 1	**System Startup Boot Process**	Describes how to use the NPS-400 system software; explains important concepts with regards to CTOPs, describes the architecture of the EZchip software from boot loader to SMP Linux, and gives an overview of I/O device drivers for the NPS-400.
Section 2	**Linux Interfaces and Customization**	Describes how to customize the Linux kernel and adapt it to specific requirements.
Section 3	**Making a Flash Image**	Describes how to build the flash image.
Section 4	**Software Update**	Describes how to use the boot loader, U-Boot, which facilitates booting from a SPI flash or the network.
Section 5	**Using the U-Boot Boot Loader**	Describes the boot loader used to load an operating system.
Section 6	**I/O Devices and Drivers**	Describes the I/O device drivers provided by the operating system.
Section 7	**SPL Command Format**	Describes the boot loader command format.
Section 8	**PCIe-related Configuration Settings**	Control the NPS-400 configuration space as a PCIe end point.

Revision History

REVISION	***DATE***	***DESCRIPTION OF MODIFICATIONS***
1.0	Dec. 2, 2015	Initial release.

Terminology and Conventions

General

The following terminology is used throughout this document:

TERM	*DESCRIPTION*
NPS	Refers to the EZchip NPS-400 network processor device and/or software simulator.
Channel	Refers to an NPS-400 device and/or simulator in the system
Control Plane Application	Refers to a customer-developed application responsible for configuration and management of the NPS device.
Control Plane CPU	Refers to the CPU on which the control plane application resides. This may be an external host CPU or the NPS CTOPs.
Data Plane Application	Refers to a customer-developed application responsible for packet processing on the NPS device.
DTB	Device Tree Blob
Host	External CPU (typically on a daughter card) that is used for controlling and monitoring the NPS-400.
initramfs	Purpose is to mount the root filesystem.
LDK	Linux Development Kit module provided with the EZchip SDK.
PCIe	PCI Express interface on the NPS-400.
SPL	Secondary Program Loader is a small binary, generated from the U-Boot source and loaded from SPI flash into NPS IMEM.
U-Boot	Universal bootloader is an open source and primary boot loader used in embedded devices.
ZOL	Zero Overhead Linux

Typographical Conventions

The following typographical conventions are used in this manual. Routine (or function/call) names are written with a parenthesis, e.g. EZdp_Create( ).

📖 Refer to the section or document referenced here for additional information on the topic.

▸ *Notes provide additional information that is not necessarily mandatory.*

Important: Contains information that is mandatory for proper confirmation and/or operation that should not be overlooked.

Contents

List of Tables and Figures

1. System Startup Boot Process

The sequence described in this chapter is the planned reference sequence for the NPS-400 SoC device. The sequence used and features supported by the software simulator and hardware emulator systems may differ.

1.1 Overview

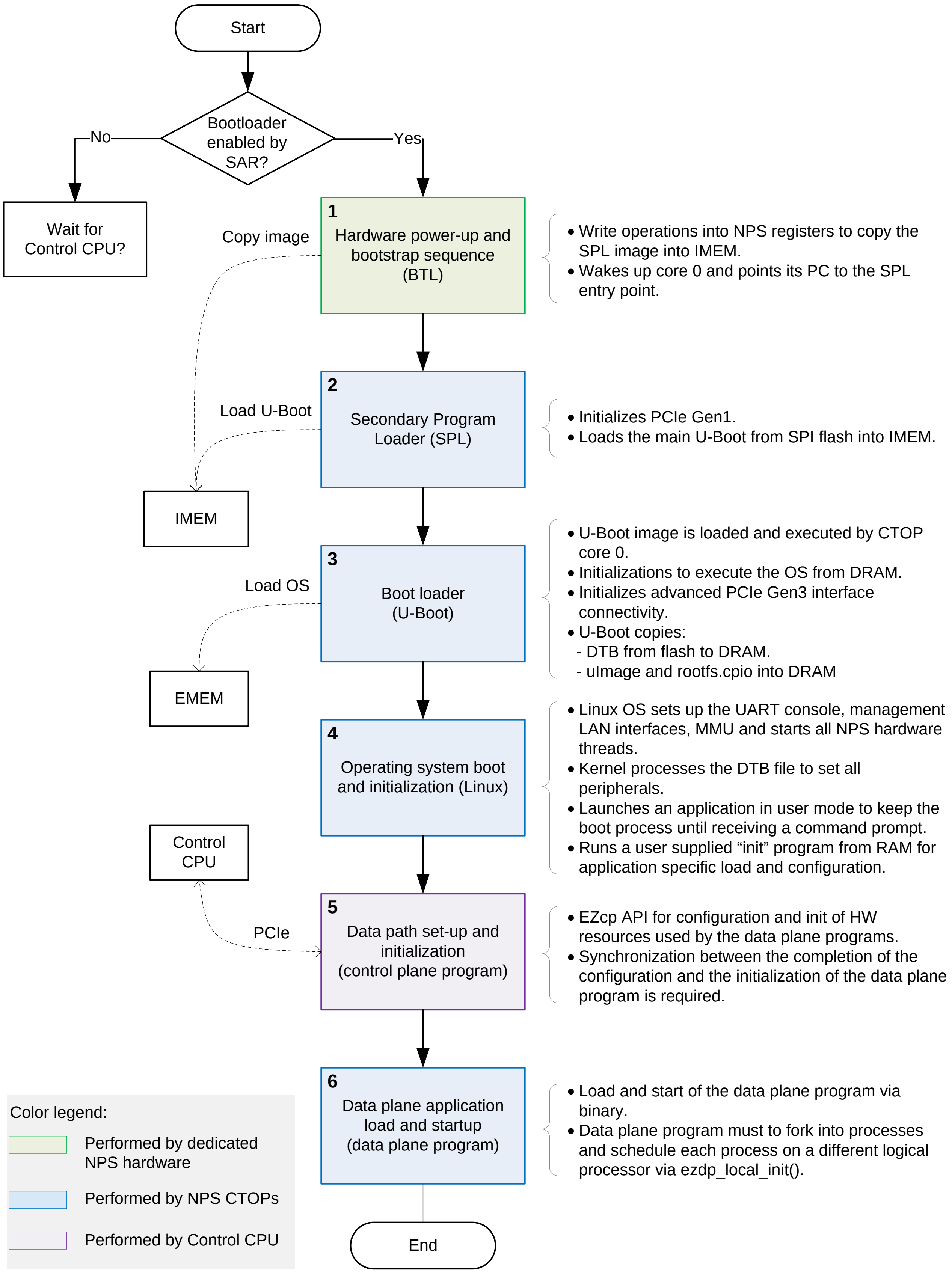
The NPS-400 software load sequence is comprised of the following main stages:

1. Hardware power-up and bootstrap sequence (BTL)
2. Secondary Program Loader (SPL)
3. Boot loader (U-Boot)
4. Operating system boot and initialization (Linux)
5. Data path set-up and initialization (control plane program)
6. Data plane application load and startup (data plane program)

Table 1. Overview of stages in NPS-400 software load sequence

NO.	STAGE	RUNS ON	HIGHLIGHTS
1	Hardware power-up and bootstrap sequence (BTL)	Dedicated NPS HW	▪ Write operations into NPS registers to copy the SPL image into IMEM. ▪ Wakes up core 0 and points its PC to the SPL entry point.
2	Secondary Program Loader (SPL)	NPS CTOPs	▪ Initializes PCIe Gen1. ▪ Loads the main U-Boot from SPI flash into IMEM.
3	Boot loader (U-Boot)	NPS CTOPs	▪ U-Boot image is loaded and executed by CTOP core 0. ▪ Initializations to execute the OS from DRAM. ▪ Initializes advanced PCIe Gen3 interface connectivity. ▪ U-Boot copies: ▪ DTB from flash to DRAM. ▪ uImage and rootfs.cpio into DRAM
4	Operating system boot and initialization (Linux)	NPS CTOPs	▪ Linux OS sets up the UART console, management LAN interfaces, MMU and starts all NPS hardware threads. ▪ Kernel processes the DTB file to set all peripherals. ▪ Launches an application in user mode to keep the boot process until receiving a command prompt. ▪ Runs a user supplied “init” program from RAM for application specific load and configuration.
5	Data path set-up and initialization (control plane program)	Host CPU	▪ EZcp API for configuration and init of HW resources used by the data plane programs. ▪ Synchronization between the completion of the configuration and the initialization of the data plane program is required.
6	Data plane application load and startup (data plane program)	NPS CTOPs	▪ Load and start of the data plane program via binary. ▪ Data plane program must to fork into processes and schedule each process on a different logical processor via ezdp_local_init().

Figure 1. Overview of stages in NPS-400 software load sequence


Start
Bootloader enabled by SAR?
No
Yes
Wait for Control CPU?
1 Hardware power-up and bootstrap sequence (BTL)
Copy image
2 Secondary Program Loader (SPL)
Load U-Boot
IMEM
3 Boot loader (U-Boot)
Load OS
EMEM
4 Operating system boot and initialization (Linux)
Control CPU
PCIe
5 Data path set-up and initialization (control plane program)
6 Data plane application load and startup (data plane program)
End
Color legend:
🟩 Performed by dedicated NPS hardware
🟦 Performed by NPS CTOPs
🟪 Performed by Control CPU

- Write operations into NPS registers to copy the SPL image into IMEM.
- Wakes up core 0 and points its PC to the SPL entry point.

- Initializes PCIe Gen1.
- Loads the main U-Boot from SPI flash into IMEM.

- U-Boot image is loaded and executed by CTOP core 0.
- Initializations to execute the OS from DRAM.
- Initializes advanced PCIe Gen3 interface connectivity.
- U-Boot copies:
  - DTB from flash to DRAM.
  - uImage and rootfs.cpio into DRAM

- Linux OS sets up the UART console, management LAN interfaces, MMU and starts all NPS hardware threads.
- Kernel processes the DTB file to set all peripherals.
- Launches an application in user mode to keep the boot process until receiving a command prompt.
- Runs a user supplied “init” program from RAM for application specific load and configuration.

- EZcp API for configuration and init of HW resources used by the data plane programs.
- Synchronization between the completion of the configuration and the initialization of the data plane program is required.

- Load and start of the data plane program via binary.
- Data plane program must to fork into processes and schedule each process on a different logical processor via ezdp_local_init().

Each of these stages is described in the sections that follow.

1.2 Hardware Power-up and Boot Sequence

Upon power-up, the NPS-400 hardware performs an automatic boot sequence described in detail in the "Power-up and Initialization" section of the *NPS-400 Architectural Specification* document.

The PCB sampled at reset parameters determine the core frequency and bootstrap mode. The bootstrap supports two main modes:

1. Hardware boot sequence reads a binary image file from the SPI flash into IMEM memory and starts core 0 of cluster 0 executing at the address the file was loaded. This feature is typically used to load an image of the SPL pre-boot software into IMEM to continue with an autonomous boot process.
2. PCIe initialization allowing a control plane program running on the host connected via PCIe to continue the boot process under the host control.

The rest of this document assumes an autonomous boot sequence (option 1 above).

The SPL is prepended with special information controlling the pre-boot phase before actually running SPL by core 0. The header format is a series of write operations into NPS-400 configuration registers, the series usually ends by copying the SPL image into IMEM, and finally it will wake up core 0 and will point its PC to the SPL entry point which is right at the beginning of the SPL image on IMEM.

📖 Refer to the *NPS-400 Hardware Reference Manual* for values sampled at reset.

1.3 Secondary Program Loader (SPL)

Secondary Program Loader (SPL) is a small binary, generated from the U-Boot source and loaded from SPI flash into NPS IMEM. Once loaded in IMEM and executed by the hardware boot sequence, SPL proceeds to initialize PCIe (Gen1). Such a small binary is important in terms of load time into IMEM since PCIe initialization should be completed before BIOS on the host will timeout on PCIe link training. Once PCIe link is established and enumeration procedure is started the SPL will load the main U-Boot from SPI flash into IMEM.

1.4 Boot Loader (U-Boot)

U-Boot (Universal Bootloader) is an open source and primary boot loader used in embedded devices. The U-Boot image is loaded from SPI Flash to IMEM and executed by the designated CTOP core (usually core 0).

Once loaded to IMEM and executed by the SPL, U-Boot performs all the initializations needed to execute the operating system from DRAM. After initializing the DRAM controllers and memory settings, U-Boot proceeds to initialize advanced PCIe interface connectivity e.g. Gen3, the UART, and any other hardware resource required for loading the operating system. An additional task performed by the U-Boot is copying of the Device Tree Blob (DTB) from flash to DRAM and updating the DTB according to U-Boot environment variables. U-Boot also copies uImage and rootfs.cpio into DRAM before the "bootm" command is invoked. This generic command uses the uImae header to know where the operating system entry point is located. In development mode DTB, uImage and rootfs.cpio can be retrieved over the network using management LAN that previously was initialized by U-Boot.

📖 U-Boot supports many additional features and options which are outside the scope of this document. Further details are available at the U-Boot web site at http://www.denx.de/wiki/U-Boot.

1.5 Operating System Boot and Initialization (Linux)

At the conclusion of the boot loader phase, the Linux operating system is handed control. Linux sets up the UART console, management LAN interfaces, MMU and starts all NPS hardware threads on all cores based on the configuration passed in the kernel command line.

One important kernel command line parameter which needs to be passed to the operating system is designation of the logical processors that will be used for data plane processing and, therefore, require isolation from interference by the operating system.

The kernel processes the DTB file that is passed by U-Boot to set all peripherals. Parameters that are passed from U-Boot include the core frequency, DRAM size, UART baud rate, active CPUs, network parameters and more. The kernel will wake up each one of the threads of each active core to be presented as a logical CPU. When all active CPUs are up, the kernel proceeds to its final steps of boot process. Once the kernel is done, it launches an application in user mode to keep the boot process until receiving a command prompt.

As part of the Linux initialization sequence, Linux typically runs a user supplied program called “init” from the RAM file system which is used to perform any application specific load and configuration.

1.6 Data Path Setup and Initialization (Control Plane Program)

Prior to the loading and initialization of data plane programs, certain configurations and initializations of hardware resources used by the data plane programs need to be completed. This task is accomplished by a control plane program via calls to the EZchip-provided EZcp control plane API.

The control program performing this phase may run either on an external host connected via PCIe, or on the NPS itself. In either case, note that the data plane program will not be able to initialize prior to the completion of this setup.

The synchronization between the completion of the configuration and the initialization of the data plane program may be performed in any means chosen by the user, such as having the control program on the host run the data plane programs on the device via a remote shell connection over the management LAN after the configuration phase is finished.

1.7 Data Plane Application Load and Startup (Data Plane Program)

Once everything is set up, loading and starting the data plane program is achieved by running the data plane program binary.

The data plane program binary may be included as part of the initial RAM file system loaded at boot time or may be downloaded to the file system at run time, i.e. by FTP protocol from an external host via the management LAN.

Regardless of the method of the binary program acquisition, starting the program is achieved by executing it, just like any other Linux user space program. If the binary is loaded at run time via an external process, care should be taken when assigning proper executable permission to the binary image.

Once executed, it is the data plane program’s responsibility to fork itself to as many processes as it requires and to schedule each process on a different logical processor by providing the logical processor number to the **ezdp_local_init()** call during the data plane program initialization.

2. Linux Interfaces and Customization

This chapter assumes that the EZdk software is available and that it has received LDK (Linux Development Kit) sources with Buildroot sources and all packages required by the NPS Buildroot default configuration.

- *LDK is a module provided with the EZchip SDK.*

2.1 Building Linux from Source

Linux may either be built through Buildroot (recommended) or directly as described below.

2.1.1 Building through Buildroot

This is the preferred method.

The first time that it is used, the Buildroot sources must be extracted from <EZdk>/ldk/sources.

From the Buildroot source directory, set the configuration by “make arceb_nps_defconfig”.

Finally compile Buildroot to create all that was configured by “make”.

After any Linux change, it can be rebuilt by “make linux” or reconfigured by “make linux-xconfig”.

2.1.2 Building Directly

For the first time that it is used, Linux sources must be extracted from <EZdk>/ldk/sources.

Set environment variables:

```
ARCH=arc
CROSS_COMPILE=“<EZdk>/ldk/toolchain/bin/arceb-linux-”
```

From the Linux source directory, set the configuration by “make nps_defconfig”.

Finally compile Linux to build all that was configured by “make”.

After any Linux change, it can be rebuilt by “make” or reconfigured by “make xconfig”.

2.2 Configuring Linux

Several kernel build options and device tree options have been added to aid with Linux setup.

Build option controls whether the core can use more than one hardware thread:

```
CONFIG_EZNPS_MTM_EXT
```

Device tree options:

```
present-cpus = <cpu list>
possible-cpus = <cpu list>
```

where cpu list format is the same as generic “isolcpus”.

Additional information on these options can be found in the EZsim appendix of the *EZide User Guide*.

2.3 Linux Boot Arguments

The NPS default build configuration has all CPUs set with timer mode of “nohz_full”. This can be changed at run time with this generic boot argument as a kernel parameter:

```
nohz_full=<cpu list>
```

Only CPUs with “nohz_full” can be used for data plane applications, in which case no operating system interference is allowed.

2.4 Controlling the Linux Initramfs Boot Process

Since NPS does not have persistent storage, initramfs is used to hold all the file system structure built by Buildroot.

To modify file system permanently, change <EZdk directory>/ldk/sources/rootfs_overlay and run “make” for Buildroot.

For a transient change, you may use FTP to get a new file for running NPS.

There is an alternative to hold the file system with root NFS through the management LAN which may be useful for development.

2.5 Enabling Zero Overhead Linux (ZOL) with the Data Plane Boot Argument

Zero Overhead Linux (ZOL) is enabled by default on all CPUs.

3. Making a Flash Image

The provided SDK includes a module named *Linux Development Kit*, hereafter referred to as LDK in this document, which includes a Buildroot tool supporting the automated build of the Linux kernel, file system and U-Boot compatible images.

3.1 Flash Layout

The hardware boot sequence upon bootstrap configuration may start executing the SPL binary header from offset 15MB or 15.5MB at flash device (or none).

Typical the first 16MB of the flash would be populated as shown:

Table 2. Content of the first 16MB of the flash (typical)

FLASH OFFSET	*IMAGE DESCRIPTION*
0x00f80000	SPL binary image prepended with header for HW boot sequence (second copy)
0x00f00000	SPL binary image prepended with header for HW boot sequence
0x00e80000	U-Boot binary image and environment.
0x00700000	Initramfs CPIO image, e.g. rootfs.cpio.
0x00100000	Linux binary image, e.g. uImage.
0x00000000	Device Tree Blob, e.g. nps.dtb.

3.2 Creating a Bootrom

LDK contains several scripts to create a flash image under name “flash” project. This project creates a 16MB raw image structured as shown in the table above (Table 2).

To create an image, extract the project archive at LDK sources. Set the environment variable:

```
EZDK_BASE = <EZdk directory>
```

Then run “make”.

For better usage of flash, we recommend to make this image in JFFS2 file system format.

This will also provide the actual size of each part of the image e.g. rootfs.cpio.

4. Software Update

4.1 Application Software

Application software, both data plane and control plane, are regular user space programs which may be stopped and started at run time. The program can be stopped in the same method as any user plane program. Application software is a regular executable which may be stored on flash, loaded via the network or written to NPS memory over PCIe from an external host.

Care should be taken to ensure that the control plane and data plane software is coordinated with regards to the use of hardware resources.

4.2 Operating System and Boot Loader

The Linux kernel binary, initial RAM file system and the U-Boot boot loader are all typically loaded from the SPI flash during boot.

Updating the SPI flash is supported while the system is operating via the Linux MTD sub-system.

The hardware boot sequence supports the loading of an alternate U-Boot boot loader image from a secondary storage in the SPI flash, allowing a fail-safe upgrade of the boot loader.

📖 For details on this feature and its usage please consult reference the “Power-up and Initialization” section of the *NPS-400 Architectural Specification* document.

The U-Boot boot loader supports scripted loading of an alternate operating system and file system images allowing a fail-safe upgrade of the operating system and initial file system.

📖 For details of this feature and its usage please consult the U-Boot documentation.

5. Using the U-Boot Boot Loader

A Boot Loader is primarily used to load an operating system. As such, the Boot Loader needs to initialize the DRAM where the operating system is executing from.

Several environment variables for U-Boot should be set and saved into persistent memory. In the NPS, this is dedicated sections on SPI flash.

One important variable is “bootcmd” which defines what U-Boot does when initialization is done and the main loop is reached. Usually it will load an operating system (uImage) into DRAM and will use its header to invoke the operating system from the correct offset on DRAM.

The “bootargs” variable is useful for passing arguments to the kernel. Actually U-Boot concatenates its value with a command line option defined in DTB.

The “bootargs” variable can be used to pass network parameters provided statically from variables “ipaddr”, “netmask” and “hostid” or alternatively from the DHCP server.

There are U-Boot commands added to support BIST running on the DRAM sub system.

6. I/O Devices and Drivers

Operating system provides several I/O device drivers:

- Management LAN
- UART Serial
- Data Plane Linux (DPL)

Management LAN

Selected by default with option CONFIG_EZCHIP_NPS_MANAGEMENT_ENET.

This 1Gb port that can serve to download a data plane application and run it, or maybe perform a software update or any other purpose one can think of.

UART Serial

This is an RS232 device used with a simple console to receive early logs even before the management LAN is available.

This is useful for early bring up of a new board where the management LAN is not yet operational.

DPL

This is needed by the data plane application to access resources that can be accessed only at privileged mode of core, i.e. operating system.

This driver provides “ioctl” for accessing auxiliary registers and also “mmap” that GDB uses to view private core memory (CMEM).

7. SPL Command Format

Boot loader has a set of the following commands:

Table 3. Boot loader commands

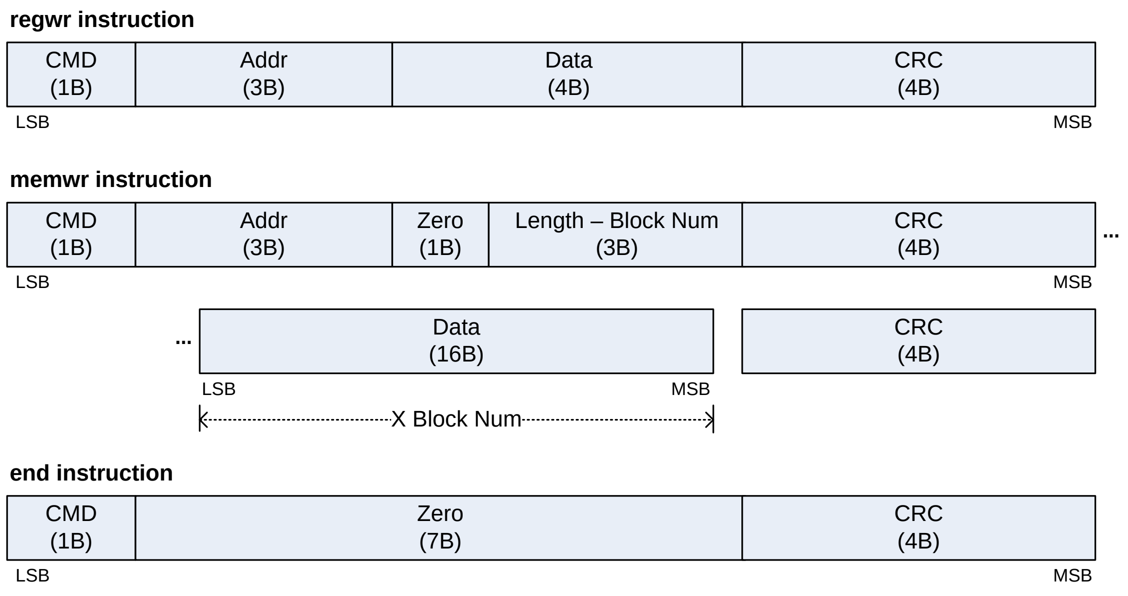
COMMAND NAME	*COMMAND ENCODING*	*DESCRIPTION*
regwr	8'b1100_0000	Write register
memwr	8'b0011_0000	Write memory block
end	8'b0000_1100	Stop
memwr + wake + end	8'b0000_0011	Sequence of: ▪ Write memory block ▪ Wake the core ▪ Stop

For the purpose of simplicity, all the instructions have the same header structure:

- 1B of command
- 7B of data
- 4B of CRC

In case of write memory block, the header will be followed by 16B blocks of data that ends with CRC32.

Figure 2. Boot loader: packet format for write memory block



Inside the LDK there is an Assembler (Python script) to convert a BTL text file into a binary header prepended to SPL.

8. PCIe-related Configuration Settings

When using NPS-400 as a smart NIC and it is seen by the root complex on a host as a network device endpoint, one can configure how it will look, i.e. its vendor ID, BAR size, capabilities that will be included and so on.