Users Manual

RZ/N1 U-Boot

System-on-Chip

Target Device RZ/N1

Renesas Electronics
www.renesas.com

R01US0296EG0106
July 5, 2019

Notice

Trademarks

- Linux® is a registered trademark or a trademark of Linus Torvalds in the United States and/or other countries.
- Other company names and product names mentioned herein are registered trademarks or trademarks of their respective owners.
- Registered trademark and trademark symbols (® and ™) are omitted in this document

Disclaimers

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics.

8. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti- crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.

9. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

10. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.

11. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

13. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

    (Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority- owned subsidiaries.

    (Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

Table of CONTENTS

1 OVERVIEW

1.1 Overview

This manual describes the U-Boot software developed for the Renesas RZ/N1D, RZ/N1S and RZ/N1L devices.

U-Boot is provided in two forms:

- Standard U-Boot providing full U-Boot functionality with a command prompt.
- U-Boot/SPL providing a small image for fast boot but limited functionality.

Both are built from the same U-Boot source code. Throughout this document, where “U-Boot” is used without the SPL suffix, we refer to the standard U-Boot described above. Where this document describes commands that can be used within U-Boot, it refers to the standard U-Boot.

1.2 Functionality

The software includes the following functionality:

- U-Boot
    - Drivers for:
        - Clock Controller
        - PinMux
        - UART
        - DDR
        - QSPI
        - NAND Flash
        - Ethernet MAC
        - 5-port Switch
        - RGMII/RMII Converters
        - SDHC
        - I2C
        - USB Host
        - USB Function
    - USB DFU support
    - SPL support

2 PROGRAMMING QSPI/NAND

2.1 Background

The RZ/N1D and RZ/N1S use a BootROM to load code into SRAM on reset. The BootROM loads an SPKG image, which is a payload along with a special header consisting of information about the length of the payload and where to copy it to. Depending on boot mode pins, the BootROM will load the SPKG from SPI Flash, NAND Flash or via USB using the DFU protocol.

The RZ/N1L does not use a BootROM, instead it starts executing code stored in SPI Flash.

All devices in the RZ/N1 family can program QSPI using the IAR Embedded Workbench IDE, however it is limited in scope, details are provided below.

If you need to program NAND Flash, or require broader support for programming SPI Flash, you can use U-Boot to do so.

2.2 IAR Embedded Workbench IDE

The IDE provides a mechanism to program flash devices called Flash Loader. This uses a special piece of code that is downloaded into SRAM on the device that performs the flash operations on behalf of the IDE.

Renesas provide an RZ/N1 Flash Loader which you can use. However, the Flash Loader has the following restrictions:

- It cannot program NAND Flash.
- It cannot program beyond the first 16MB of Flash.
- It cannot program SPI Flash connected to QSPI2 (RZ/N1S).

Any data that you require to be programed into SPI Flash must either be linked into an existing program, or be provided as an ELF executable file where the addresses correspond to the QSPI memory-mapped address space, i.e. a program built to run directly from QSPI, rather than be loaded by a BootROM. It is not possible to program arbitrary binary files into SPI Flash.

2.3 U-Boot for RZ/N1L

In order to use U-Boot to program SPI Flash or NAND Flash, you could use the IAR Embedded Workbench IDE to program U-Boot into SRAM, and then manually fix the Program Counter and Stack Pointer to match U-Boot's vector table. However, you need another program to start execution.

Alternatively, use the RZ/N1L BootStrap project to program QSPI with boot-strap code that will copy a payload from QSPI into SRAM and run it.

2.4 U-Boot for RZ/N1D and RZ/N1S

This section provides instructions to program QSPI or NAND Flash on a new board, using U-Boot. You will need a Linux PC or virtual machine for this.

Since the RZ/N1 BootROM can only download an SPKG payload into SRAM, we need to download code that can then program QSPI or NAND Flash. The steps needed are:

- Set the boot mode pins to Boot-from-USB and reset the board.
- Using dfu-util on your host PC, download an SPKG payload into SRAM.
- From the U-Boot prompt, run dfu
- Using dfu-util on your host PC, download your application into QSPI or NAND.

Since the BootROM will only load an SPKG from SPI Flash or NAND Flash, your application must be converted into an SPKG format.

1. Create your application that you want to run on boot. Note that the BootROM will load this code into SRAM (e.g. address 0x04000000) and run it from there, so your code entry point is in the SRAM, not QSPI.
2. Convert your application binary into an SPKG image. The RZ/N1 version of U-Boot includes a tool to create an SPKG from a binary file. Here's an example, you might use a different load_address depending on what you are doing, this address is the default for U-Boot. The load_address is also the code entry point. Keep the other options the same if using QSPI, otherwise set them according to your NAND flash.

```
tools/rzn1/spkg_utility –i  YOUR_APP.BIN \
     -o  YOUR_APP.SPKG \
     --padding 64K --load_address 0x200a0000 \
     --nand_ecc_enable --nand_ecc_blksize 1 \
     --nand_ecc_scheme 1 --nand_bytes_per_ecc_block 7 \
     --add_dummy_blp
```

3. Install the 'dfu-util' package, e.g. `sudo apt-get install dfu-util`. Note that the time outs in the standard dfu-util are not enough for Micron N25Q64 devices, see section 3 for details of building dfu-util to work around this.
4. Set your board boot mode pins to boot from USB device (DFU). The boot mode is read on reset from GPIO[78:77]. Normally, your board will have switches that activate pull-up or pull-down resistors on these pins. Reset the board, the RZ/N1 serial terminal should output:

```
** BOOTLOADER STAGE0 for RZN1 **
Boot source: USB
```

5. Download U-Boot to SRAM. On your Linux PC run:

```
sudo dfu-util -D u-boot-<board>.bin.spkg
```

6. U-Boot should run and the RZ/N1 serial terminal will present you with a prompt. Run:

```
dfu
```

7. On your Linux PC run:

```
sudo dfu-util  -a  'sf_spl' –D  YOUR_APP.SPKG
```

8. Wait until it completes, the RZ/N1 serial terminal will prompt you to press Ctrl-C when done.
9. Set your board boot mode pins to boot from QSPI. When you now reset the board, the BootROM should load your SPKG and run it.

3 HOST TOOLS

Renesas recommend using the open source ‘dfu-util’ host application to use DFU.

3.1 dfu-util for Linux

Renesas have made a few small modifications to dfu-util. The first allows dfu-util to terminate after uploading the requested number of bytes, instead of continuing to load until RZ/N1 has read all of the data from the specified DFU target. The second increases the default timeout from 5 seconds to 10 seconds. This increase in time is required as it is the time U-Boot takes to erase and write the amount of data that can fit into the DFU data buffer (size specified by CONFIG_SYS_DFU_DATA_BUF_SIZE). Without this change, writing to some slow QSPI devices (e.g. Micron N25Q64) will timeout.

The following steps on your host PC will download dfu-util and apply the patches. The software package was tested using v0.9:

```
git clone https://git.code.sf.net/p/dfu-util/dfu-util
cd dfu-util
git checkout -b rzn1 v0.9
git am ${RELEASE_DIR}/dfu-util/patches/00*.patch
```

You can build and install the code using:

```
sudo apt install autoconf libusb-1.0-0-dev
./autogen.sh
./configure
make
sudo make install
```

3.2 dfu-util for Microsoft Windows

The following is a short summary on how to install dfu-util for Windows, for more details see Reference 1.

3.2.1 Start the DFU target

Connect your RZ/N1 board to your Windows PC using the USB cable, and reset your board in “Boot from USB” mode (i.e. using DFU).

3.2.2 Register the USB device

You have to register the USB Download Gadget device with the libusbk driver, you only need to do this once. On your Windows PC:

- Download Zadig from https://github.com/pbatard/libwdi/releases/download/b721/zadig-2.4.exe
- Launch Zadig, select "List All Devices" from the Options menu.
- Select "RZN1 DFU" from the pull down list.
- Select "libusbK" from the Target Driver list (to the right of the green arrow).
- Click the "Install Driver" button, this will take some time.
- Quit Zadig

3.2.3 Download dfu-util

Download the Windows version of dfu-util from https://sourceforge.net/projects/dfu-util/files and unpack the zip file. No installation is required, where this document tells you to run dfu-util, simply run the dfu-util-static.exe Windows executable from a Windows command prompt.

4 BUILD SETUP

4.1 Development Environment

This software requires a host PC with internet access in order to cross-compile the software. The software package has been tested using a host PC running Ubuntu 16.04 LTS.

The instructions assume the `RELEASE_DIR` environment variable is set to a directory containing this release installed on your host PC.

4.2 Toolchain

The software package has been tested with the 64-bit version of the Linaro 2017.02 (gcc 6.3) toolchain. The following steps will download and install the toolchain:

```
wget
    https://releases.linaro.org/components/toolchain/binaries/6.3-2017.02/
    arm-linux-gnueabihf/gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabih
    f.tar.xz
sudo tar xf gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf.tar.xz -C
    /usr/share
export
    PATH=/usr/share/gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf/bi
    n:$PATH
export CROSS_COMPILE="arm-linux-gnueabihf-"
export ARCH=arm
```

If using a 32-bit OS, please use the 32-bit version of the toolchain that can be downloaded from `https://releases.linaro.org/components/toolchain/binaries/6.3-2017.02/arm-linux-gnueabihf/gcc-linaro-6.3.1-2017.02-i686_arm-linux-gnueabihf.tar.xz`

4.3 Additional Tools

The commands used to configure and build the software require several packages to be installed. The following step will download and install the packages:

```
sudo apt-get install -y --force-yes --fix-missing build-essential
    libncurses5-dev u-boot-tools gettext bison flex libusb-1.0-0-dev
```

4.4 U-Boot

4.4.1 Setup

On your host PC, the following command will download the RZ/N1 branch for U-Boot, including all commit history. Note that it will result in a large download

```
git clone http://git.denx.de/u-boot.git
cd u-boot
```

Checkout a branch based on the 2017.01 version:

```
git checkout -b rzn1 v2017.01
```

Set the BSP version to according to the release, for example:

```
BSP_VERSION=v1.5.3
```

Fetch the RZ/N1 branch and merge it in:

```
git remote add renesas-rz https://github.com/renesas-rz/rzn1_u-boot.git
git fetch --tags renesas-rz
git merge rzn1-public-$BSP_VERSION
```

4.4.2 Build

To setup the configuration for the Renesas RZ/N1D-DB Board, run:

```
make rzn1d400-db_config
```

To setup the configuration for the Renesas RZ/N1S-DB Board, run:

```
make rzn1s324-db_config
```

To setup the configuration for the Renesas RZ/N1S IO-Link Board, run:

```
make rzn1s-io-link_config
```

To setup the configuration for the Renesas RZ/N1L-DB Board, run:

```
make rzn1l-db_config
```

To build U-Boot, run:

```
make
```

Once complete, the Elf image is stored as `u-boot`. The image is built to execute from internal SRAM on the RZ/N1 devices, see section 5.2 for details. You can download this file using a debugger.

5 U-BOOT

The U-Boot port uses standard mechanisms to configure the build for your RZ/N1 board, i.e.

- The RZ/N1D-DB board configuration is in **configs/rzn1d400-db_defconfig**
- Additional configuration not handled by Kconfig is in **include/configs/rzn1d400-db.h**

These files include a common RZ/N1 device configuration file, **include/configs/rzn1-common.h,** that provides common configuration settings and a few sensible defaults that should apply to most RZ/N1 boards. Where descriptions in this document indicate that the configuration is controlled by #define <C_SYMBOL>, it means the #define is in the header file. If the configuration is controlled via C_SYMBOL=<VAL>, it means the definition is in the Kconfig file.

5.1 Help

The majority of commands have built-in help when built with long help. At the U-Boot prompt you can simply type 'help', or 'help *<command>*'. For example, 'help tftpboot'. U-Boot also recognises '?' instead of help and will match shorter text as long as it is unique.

5.2 Memory Use

U-Boot requires quite a lot of memory to support functions such as SD cards, LCD displays, NAND, etc. Since many devices that support U-Boot have a very small amount of SRAM, U-Boot normally initialises basic functionality, sets up DDR, then relocates itself to the top of DDR and continues execution from DDR. For this reason, U-Boot has an initial stack and heap that is used before relocation, and the normal stack and heap used after relocation.

Since some of the RZ/N1 devices do not support DDR, and all have large on-chip SRAM available, Renesas have implemented changes so that U-Boot does not relocate to DDR. However, the concept of initial stack and heap persists. This document assumes that relocation to DDR is disabled, using:

```
#define CONFIG_SYS_STAY_IN_SRAM
```

5.2.1 Pre-relocation

The start of the U-Boot image and code entry point is in SRAM and is defined by:

```
#define CONFIG_SYS_TEXT_BASE          0x200a0000
```

The initial stack, which is very small, will be located with:

```
#define CONFIG_SYS_INIT_SP_ADDR      (CONFIG_SYS_TEXT_BASE + (320 * 1024) - 4)
```

The initial heap sits immediately above the stack. The initial heap defaults to 1KB but may be increased with:

```
CONFIG_SYS_MALLOC_F_LEN=0x800
```

5.2.2 Post-relocation

Once U-Boot passes the point where it would relocate to DDR, all data is placed beneath CONFIG_SYS_SRAM_BASE + CONFIG_SYS_SRAM_SIZE, as per the following table.

Use	Size	Address
Top of SRAM		CONFIG_SYS_SRAM_BASE+ CONFIG_SYS_SRAM_SIZE
MMU TLB Page Table	16KiB	
Environment variables	CONFIG_ENV_SIZE	

Heap	CONFIG_SYS_MALLOC_LEN	
Board Info	sizeof(gd_t), i.e. 0x80	
Stacks		

Table 1: U-Boot Memory Use

If U-Boot is built with `CONFIG_CMD_BDI` defined, you can run the `bdinfo` command to display information about the addresses used.

Also note that U-Boot provides a default address that is used by several commands to store data such as images downloaded via serial. This is specified by:

```
#define CONFIG_SYS_LOAD_ADDR 0x80080000
```

For some commands, but not all, this address is overwritten by the `loadaddr` environment variable. In particular, this symbol is used by U-Boot/SPL as environment variables are not available.

5.3 System setup

Code associated with system setup is split into RZ/N1 device specific and board specific parts.

The device specific code can be found in arch/arm/cpu/armv7/rzn1. Board specific code can be found under board/<board-provider>/<board-name>. For example, the Renesas RZ/N1D-DB board code can be found in board/renesas/rzn1d400-db.

5.3.1 Clocks

U-Boot only enables the clocks for the peripherals it uses. UART clocks are enabled in the arch_cpu_init() function and USB clocks are controlled in the rzn1_usb_init() function, both are in arch/arm/cpu/armv7/rzn1/cpu.c. Board specific clocks such as QSPI, SDHC, I2C, and GPIOs are enabled in the board's board_init() function. USB clocks are indirectly controlled via the board's board_usb_init() function which is called when the user runs commands such as dfu and usb start. U-Boot does not turn off any clocks before starting the OS.

Please note that the RZ/N1 devices have most peripheral clocks enabled by default. Depending on your application, significant power savings can be made by turning off the clocks for unused peripherals and interfaces. This can be done by defining the RZN1_TURN_OFF_CLOCKS symbol in the board configuration file. However, be aware that disabling all clocks that are not used by U-Boot may cause 3rd party software to fail if it does not enable the clocks it requires.

5.3.2 Pin Multiplexing

Renesas provides an RZ/N1 PinMux App developer tool that allows you to configure the pin functions, drive strengths and pull up/down resistors. This tool can output a Device Tree file and a C header file with this information. The C header file defines a generic `rzn1_board_pinmux()` function and a table containing the pin settings. Note that these generated files also include virtual pin configuration for the MDIO masters. See section 5.9.2 for further details.

Board specific code in U-Boot uses the C header file generated by the PinMux App to configure the pinmux.

5.3.3 Caches

By default, U-Boot enables the instruction and data caches. To disable them, specify:

```
#define CONFIG_SYS_ICACHE_OFF
#define CONFIG_SYS_DCACHE_OFF
```

5.3.4 DDR Initialisation

For RZ/N1D boards, U-Boot initialises the DDR Controller with values that are defined in a header file. The header file is generated using a Cadence tcl script that generates it from Denali SOMA files for the DDR devices. U-Boot is provided with standard DDR3 Controller values for JEDEC 1333MHz devices.

To enable the DDR Controller driver, specify:

```
#define CONFIG_CADENCE_DDR_CTRL
```

By default, U-Boot sets the controller to 16-bit wide data. To select 8-bit wide data, specify the following in the configuration:

```
#define CONFIG_CADENCE_DDR_CTRL_8BIT_WIDTH
```

If using DDR with ECC, by default U-Boot does not enable ECC. To enable it, specify the following in the configuration:

```
#define CONFIG_CADENCE_DDR_CTRL_ENABLE_ECC
```

Note: When enabling ECC, the controller must also be set to 8-bit wide data, and the size of the useable DDR is halved.

5.4 UART Driver

5.4.1 Configuration

The driver is enabled with:

```
Device Drivers  --->
  Serial drivers  --->
    -*- NS16550 UART or compatible
```

The driver is initialised using DT. The driver additionally requires the following settings:

```
#define CONFIG_SYS_NS16550_CLK              47619047
#define CONFIG_SYS_NS16550_MEM32
```

The CONFIG_SYS_NS16550_CLK definition is also used to program the PLL clock dividers for the UART blocks, therefore depending on your use of the UARTs you may require different PLL settings.

The driver supports baud rates of 57600, 115200 and 1000000. The default baud rate is selected by:

```
#define CONFIG_BAUDRATE 115200
```

Since the full UART driver reads setup information from DT, it may be hard to debug errors when porting to new boards. To assist with this use the debug UART driver, which is enabled with:

```
Device Drivers  --->
  Serial drivers  --->
    [*] Enable an early debug UART for debugging
          Select which UART will provide the debug UART (ns16550)  --->
    (0x40060000) Base address of UART
    (20000000) UART input clock
    (2)   UART register shift
```

When using the debug UART, the baud rate is not set. It remains at that used by the BootROM, i.e. 115200.

5.5 ARM Timer Driver

5.5.1 Configuration

On the RZ/N1D and RZ/N1D devices, the following option enables the ARM Architected timer driver for the Cortex A7. On the RZ/N1L device, the option enables the ARM SysTick timer driver for the Cortex M3:

```
#define CONFIG_SYS_ARCH_TIMER
#define CONFIG_SYS_HZ_CLOCK          6250000
#define CONFIG_SYS_CLK_FREQ          CONFIG_SYS_HZ_CLOCK
```

The driver does not use DT as there is no other configuration information for the timer. The timer number of clock ticks is accessed using a special instruction.

5.5.2 Commands

U-Boot and drivers use the timer in a number of places. Additionally, the timer can be used via the sleep and timer commands.

To enable the sleep and timer commands, select:

```
Command line interface  --->
  Misc commands  --->
    [*] sleep
    [*] timer
```

5.6 NAND Flash Controller Driver

5.6.1 Restrictions

Due to the need for DFU to store an entire erase block before writing to NAND, and the limited on-chip SRAM memory available, devices with larger erase blocks than 128KB are not supported. It may be possible to support larger erase blocks, but it would require careful consideration of memory use.

5.6.2 Configuration

The driver requires the following to be defined:

```
#define CONFIG_SYS_NAND_CLOCK        83333333
#define CONFIG_SYS_NAND_SELF_INIT
#define CONFIG_SYS_NAND_ONFI_DETECTION
```

The CONFIG_SYS_NAND_CLOCK definition is also used to program the PLL clock divider for the NAND Controller block.

The driver does not use DT.

5.6.3 ECC

The choice of ECC scheme is limited by the size of the OOB/spare region available per NAND page. The OOB area is used to store the ECC signature, along with:

- Bad block markers (2 bytes)
- File system metadata

The main two file systems used for NAND Flash are JFFS2 and UBIFS. UBIFS does not use the OOB data for metadata, whereas JFFS2 uses 8 bytes per page for clean markers.

For a 512 byte codeword, 13 bits are required for each bit of BCH error correction. Therefore, the required bytes for the ECC signature per 512 codeword is:

- 7 bytes for BCH4, calculated from (13 * 4 + 7) / 8.
- 13 bytes for BCH8, calculated from (13 * 8 + 7) / 8.
- 26 bytes for BCH16, calculated from (13 * 16 + 7) / 8.

However, the number of bytes per 512 byte codeword used by the Cadence NFC varies, depending on the IP configuration settings. For the configuration settings used by the RZ/N1 device, it uses:

- 7 bytes for BCH4
- 14 bytes for BCH8.
- 28 bytes for BCH16.
- 42 bytes for BCH24.
- 56 bytes for BCH32.

As an example, a typical NAND Flash device has a page size of 2048, and 64 bytes of OOB data. Therefore, the number of bytes of OOB used when using BCH4 with JFFS2 is:

= “BCH bytes per 512 byte codeword” * (“Page Size” / 512) + 2 + “File system metadata”

= (7 * (2048 / 512)) + 2 + 8

= 38 bytes of OOB used

Note: If no bytes are reserved for the file system metadata (e.g. when using UBIFS), it would be possible to use BCH8 on the above device.

The Cadence NFC driver reads the device parameters and then calculates the maximum strength BCH that can be used. In order to do this, the driver needs to know how many bytes per page in the OOB area are required for file system metadata. This is specified with:

```
#define RZN1_NAND_OOB_FS_BYTES       8 /* JFFS2 */
```

Use of spare OOB

The ECC covers bit flips in the main area and the BCH bytes. Since the OOB area has the same attributes as the main area, this is essential for useful operation. However, spare OOB bytes, i.e. those not used by for ECC, do not have any error detection or correction.

Therefore, depending on the use of the spare OOB and the devices characteristics, it may be advisable to avoid the use of spare OOB. This is one reason why many modern Linux systems use UBIFS instead of JFFS2.

On-die ECC

Some older NAND Flash devices can generate and use their own ECC hardware, i.e. they use parts of the OOB by themselves to perform error detection and correction. However, there are limitations with using an on-die ECC:

- Since the NAND Flash device is not aware of other uses of the OOB, e.g. a file system, the device typically does not use all of the OOB. Therefore, it may be possible to use a stronger ECC than the on-die ECC.
- On-die ECC is typically slower than can be achieved with the Cadence NFC.

Therefore, it would be preferable to disable the on-die ECC.

However, some of these devices enable the on-die ECC after a cold reset. Since the BootROM does not have the ability to identify these devices and disable the on-die ECC, it must be used with ECC disabled in the Cadence NFC.

U-Boot has been modified so that it detects devices that enable on-die ECC by default, and disables ECC in the Cadence NFC for them. Note that it is possible to use different ECC schemes for different areas of the NAND Flash device, but this is outside the scope of the delivered U-Boot functionality.

5.6.4 Commands

The built-in help shows the following:

```
nand - NAND sub-system

Usage:
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
    read/write 'size' bytes starting at offset 'off'
    to/from memory address 'addr', skipping bad blocks.
nand erase [clean] [off size] - erase 'size' bytes from
    offset 'off' (entire device if not specified)
nand bad - show bad blocks
nand dump[.oob] off - dump page
nand scrub - really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset (UNSAFE)
```

Further information is available at http://www.denx.de/wiki/DULG/UBootCmdGroupNand.

Note: The ‘nand biterr’ command is not implemented in U-Boot at all.

5.7 QSPI Controller Driver

5.7.1 Configuration

The driver is enabled using:

```
Device Drivers  --->
  SPI Support  --->
    -*- Enable Driver Model for SPI drivers
    -*-   Cadence QSPI driver
    -*-     Enable direct (memory mapped) mode for Cadence QSPI driver
```

The driver is initialised using DT. The driver additionally requires the following settings:

```
#define CONFIG_SPI_REGISTER_FLASH
```

U-Boot reads the manufacturer and device information from the SPI Flash device. U-Boot contains a table detailing the supported modes, erase page size, etc for every SPI Flash device that is supported.

Clocks

U-Boot uses the following symbols to set the output SPI clock speed (in Hz) used by all SPI Controller drivers:

```
#define CONFIG_SF_DEFAULT_SPEED      62500000
```

When you run the ‘sf probe’ command, if you do not specify the SPI clock speed, U-Boot will use the value defined by CONFIG_SF_DEFAULT_SPEED.

The SPI clock rate used to read and write U-Boot environment variables is specified in Hz by:

```
#define CONFIG_ENV_SPI_MAX_HZ        62500000
```

The QSPI Controller driver will then attempt to meet the output SPI clock speed specified to ‘sf probe’ by programming the baud rate divisor in the QSPI Configuration Register to the highest clock rate that is below the target rate. In order to calculate this, the QSPI Controller’s Host reference clock (ref_clk) must be specified by:

```
#define CONFIG_CQSPI_REF_CLK         250000000
```

The QSPI Controller can set the output SPI clock to a maximum of half the Host reference clock. In addition, the Controller driver programs delays that are based on the Host reference clock and the output SPI clock, please see QSPI Device Delay Register documentation. Also note that the driver will attempt to optimise the Read Capture Delay register settings to ensure QSPI works.

Mode

The QSPI Controller can be set up to either quad output fast mode or quad IO fast mode. U-Boot will automatically use the widest command supported by the QSPI device. When using quad modes, U-Boot defaults to quad output fast mode as the default number of dummy cycles used by different manufacturers is typically the same. It is possible to force U-Boot to use quad IO mode by specifying:

```
#define CONFIG_SPI_FLASH_READ_QUAD_CMD      CMD_READ_QUAD_IO_FAST
```

Dummy Cycles

The number of dummy cycles required by the QSPI device is specified using:

```
#define CONFIG_SPI_FLASH_DUMMY_CYCLES       6
```

This must match the number of dummy cycles that the QSPI device defaults to. U-Boot does not change register settings in the QSPI device to alter the number of dummy cycles in use.

Chip Select

The QSPI Controller outputs four chip select pins. By default, each pin selects an individual SPI Flash device. However, the chip select pins can be connected to an external chip select decoder to access more devices. This is specified by:

```
#define CONFIG_CQSPI_DECODER          1
```

If using an external decoder, the QSPI Controller needs timing information for switching the chip selects. These timings are specified in the DT.

5.7.2 Commands

The SPI flash commands are enabled with:

```
Command line interface  --->
  Device access commands  --->
    -*- sf
```

5.7.3 Use

Prior to using any commands to read, write or erase SPI flash, you must probe the SPI Flash device that you want to access. All other SPI Flash commands operate on that device. For example, to probe the default QSPI0, CS0:

```
sf probe
```

On RZ/N1S you can probe QSPI1, CS0:

```
sf probe 1:0
```

To erase SPI flash blocks:

```
sf erase <sf-offset> <size-in-bytes>
```

You can also add a '+' before the size to round up the size to whole erase blocks.

To write to SPI flash:

```
sf write <source-addr> <sf-offset> <size-in-bytes>
```

An alternative to erasing and writing is to update the SPI Flash, this command only erases and writes blocks if the data is different:

```
sf update <source-addr> <sf-offset> <size-in-bytes>
```

To read from SPI flash:

```
sf read <dest-addr> <sf-offset> <size-in-bytes>
```

5.8 USB Device – DFU Support

U-Boot supports uploading code using the DFU (Device Firmware Upgrade) protocol.

The RZ/N1 driver implements a single endpoint, as DFU doesn't use any of the other Bulk endpoints. Incidentally, since endpoint zero doesn't use DMA, the DMA function of this IP block is not supported in U-Boot.

Standard U-Boot only supports DFU on either SDRAM or NAND, i.e. a single U-Boot image can only program one of these targets via DFU. Renesas have added support for accessing the SPI Flash devices from DFU, and have implemented a "DFU Extended" mode that allows access to multiple device types from a single DFU session. This extended mode means that a single U-Boot image can be used to program SDRAM, NAND and SPI Flash.

5.8.1 Configuration

The main activation flag in the configuration file is shown below It will activate all the other necessary dependencies to activate (or deactivate) the USB Device block.

```
CONFIG_USB_GADGET=y
```

Also needed is the directive to include the 'extended' DFU mode:

```
CONFIG_CMD_DFU_EXT=y
```

This line tells the USB host that the board doesn't need any power, as it is self-powered.

```
#define CONFIG_USB_GADGET_VBUS_DRAW  0
```

When DFU is used to program NAND or QSPI, DFU needs to write data in blocks that are at least as big as an erase block for the flash memory. It has no impact if you are using DFU to write to SDRAM. This size is specified by:

```
#define CONFIG_SYS_DFU_DATA_BUF_SIZE (128*1024)
```

Note: The size of the erase block on existing modern NAND Flash devices can be 512KB. Due to the size constraints of on-chip SRAM, devices with larger erase blocks are not supported.

The following three constants define how the peripheral will appear to the host. These are Renesas USB IDs, and might be subject to licence when used outside a development and evaluation environment as on the current platforms.

```
CONFIG_G_DNL_VENDOR_NUM=0x045b
CONFIG_G_DNL_PRODUCT_NUM=0x0239
CONFIG_G_DNL_MANUFACTURER="Renesas Electronics"
```

The following declarations specifies which drivers to include for devices accessible by DFU.

Note: The RAM device is specified here but should not be included in products that want to ensure tight security using the signed package system. Consider the RAM device as debug only.

```
CONFIG_DFU_RAM=y
CONFIG_DFU_NAND=y
CONFIG_DFU_SF=y
```

This leaves the main declaration that specifies the DFU 'targets' that will be accessible from the host using dfu-utils.

The first word corresponds to the target device, and can either be 'nand', 'sf' for the QSPI serial flash, or 'ram'.

The QSPI device can target a specific bus number and CS line by specifying 'sf X:Y' where X is the bus number and Y is the CS line. In the case of RZ/N1D the bus number is always 0, whereas on RZ/N1S QSPI0 is assigned bus number 0, and QSPI1 is assigned bus number 1.

The next argument on the line correspond to the name of the DFU device, that name will be the one listed by dfu-utils when probing the USB device from the host.

The remaining arguments are identical to the original 'non-extended' DFU specification for U-Boot; they describe hardcoded offsets and size.

An example configuration is shown below. Several DFU targets have been created that reflect typical use in a final product. The minimum size of NAND targets has been set to 1MB such that is larger than the size of typical NAND erase blocks. The minimum size of SPI Flash targets has been set to 128KB such that is larger than the size of typical SPI Flash erase blocks.

```
#define DFU_EXT_INFO \
    "dfu_ext_info=" \
    "nand n_spl raw 0 100000;" \
    "nand n_rpkgt raw 100000 100000;" \
    "nand n_uboot raw 200000 100000;" \
    "nand n_env raw 300000 100000;" \
    "nand n_cm3 raw 400000 200000;" \
    "nand n_dtb1 raw 600000 100000;" \
    "nand n_kernel1 raw 700000 1000000;" \
    "nand n_dtb2 raw 1700000 100000;" \
    "nand n_kernel2 raw 1800000 1000000;" \
    "nand n_data raw 2800000 0;" \
    "sf sf_spl raw 0 10000;" \
    "sf sf_rpkgt raw 10000 10000;" \
    "sf sf_uboot raw 20000 80000;" \
    "sf sf_env raw a0000 10000;" \
    "sf sf_dtb raw b0000 20000;" \
    "sf sf_cm3 raw d0000 100000;" \
    "sf sf_kernel raw 1d0000 600000;" \
    "sf sf_data raw 7d0000 0;" \
    "sf sf_vxworks raw d0000 600000;" \
    "ram r_kernel ram 80008000 D80000;" \
    "ram r_vxworks ram 80008000 D80000\0"
```

5.8.2 QSPI SPI Flash

WARNING: Download to some slow QSPI devices using pre-built dfu-utils may fail due to hardcoded timeouts in the host tool. Please use the modified dfu-util detailed in section 3.

The QSPI DFU Driver has been optimized to minimize the amount of erase cycles per block. The main reason is that the erase cycle is very slow on this kind of device, but it also prevents wear on the flash.

- The driver detects that a block is already erased and will not perform an erase cycle before writing if it's the case.
- The driver also detects when the data written only clears bits of the data already in the flash, and will not perform an erase cycle in that case.

5.8.3 Use

When programming using DFU, it is imperative that data sent to RZ/N1 is a multiple of the erase block size of the targeted flash device. The reason is that U-Boot caches the data being sent, and only when it has received a full erase block's worth of data will it send the erase and program commands to the flash device.

From your host PC, run this command:

```
$ dfu-util -l
dfu-util 0.9
...
Found DFU: [045b:0239] ... alt=10, name="r_vxworks", serial="UNKNOWN"
Found DFU: [045b:0239] ... alt=9, name="r_kernel", serial="UNKNOWN"
Found DFU: [045b:0239] ... alt=8, name="sf_vxworks", serial="UNKNOWN"
Found DFU: [045b:0239] ... alt=7, name="sf_data", serial="UNKNOWN"
Found DFU: [045b:0239] ... alt=6, name="sf_kernel", serial="UNKNOWN"
Found DFU: [045b:0239] ... alt=5, name="sf_cm3", serial="UNKNOWN"
Found DFU: [045b:0239] ... alt=4, name="sf_dtb", serial="UNKNOWN"
...
```

The ‘name=’ entries listed here are the names that are specified in the U-Boot dfu_ext_info environment variable. The different names allow the host to download and upload a file to that specific target device/location.

Assuming your board uses a section called r_kernel that maps to DDR, to upload a host file to that section of memory in the DDR, you can now use:

```
$ dfu-util -a r_kernel -D CA7_Image_filename
dfu-util 0.9
...
Opening DFU capable USB device...
ID 045b:0239
Run-time device DFU version 0110
Claiming USB DFU Interface...
Setting Alternate Setting #9 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0110
Device returned transfer size 4096
Copying data from PC to DFU device
Download       [=========================] 100%     3572191 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

Your file is now in DDR at 0x80008000.

5.9 Ethernet MAC Driver

5.9.1 Configuration

The RZ/N1 has a complex networking sub-system that includes multiple Synopsys DesignWare Gb Ethernet MACs (GMACs), a 5-port switch, RGMII/RMII Converters, other Ethernet based hardware accelerators, and associated control. The following diagram gives an overview of the RZ/N1 Ethernet parts as used in U-Boot.

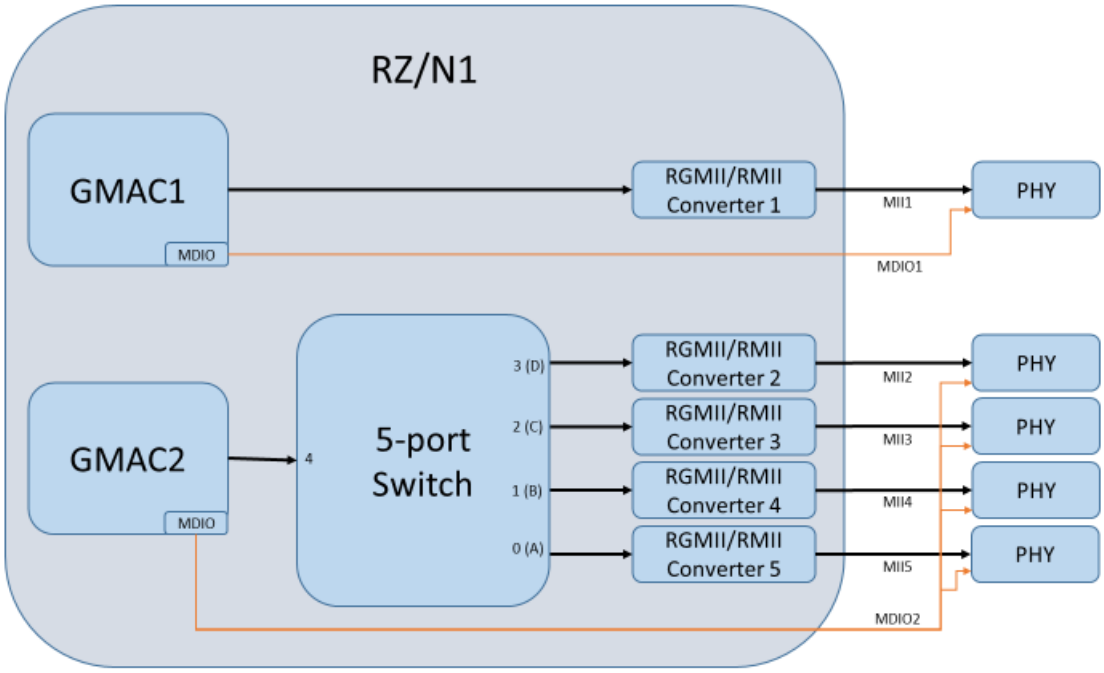

Figure 1 RZ/N1 Ethernet overview

The Synopsys DesignWare GMAC driver is enabled using:

```
Device Drivers  --->
  [ ] Enable Driver Model for Ethernet drivers
  [*] Network device support  --->
    [*]   Synopsys Designware Ethernet MAC
```

However, the driver additionally requires the following setting:

```
#define CONFIG_DW_ALTDESCRIPTOR
```

The driver as used does not use DT to obtain information about the register locations, etc, and the PHYs used.

Additionally, the PHY must be configured; this may depend on the PHY used:

```
#define CONFIG_MII
#define CONFIG_PHY_GIGE
#define CONFIG_PHY_RESET_DELAY       1000          /* in usec */
```

U-Boot supports IEEE 802.3-2002 clause 22 PHYs by default. However, for Gigabit PHYs a PHY driver for the manufacturer needs to be enabled, for example:

```
#define CONFIG_PHY_MARVELL
```

You can control which PHY to use with the first GMAC by specifying the PHY address:

```
#define CONFIG_PHY_ADDR              8
```

The RZ/N1 version of U-Boot sets up the 5-port Switch such that GMAC2 is connected to all switch ports, i.e. is an unmanaged switch. However, U-Boot does not allow you to use more than one PHY per Ethernet MAC, so you must change the PHY address to use a different MII port.

If you enable a second GMAC, specify the default PHY address using:

```
#define CONFIG_HAS_ETH1
#define CONFIG_PHY1_ADDR             4
```

This can be overridden by specifying the PHY address in the switch_phy_addr environment variable, e.g.

```
setenv switch_phy_addr 10
saveenv
```

5.9.2 Porting to different boards

The GMACs are standard Synopsys Gb MACs. There are several other IP blocks that can act as a MAC, but these are specific to industrial networking protocols so U-Boot does not use these. The RGMII/RMII Converters allow MII, RMII or RGMII to be used on external PHY interfaces. They also handle different reference clock setups. The 5-port switch is essentially an unmanaged Ethernet switch, but U-Boot only allows connection to a single PHY. The RIN Engine is not shown on the diagram, but contains registers that need to be set up to connect the IP blocks as per the above diagram.

To aid Ethernet setup, U-Boot includes several functions in:

`arch/arm/cpu/armv7/rzn1/eth.c`.

The RZ/N1D-DB board code provides a good template for calling these.

Serial Management Interface

In the RZ/N1 device, several IP blocks can be used to access SMI (i.e. MDIO/MDC). The SMI outputs from these IP blocks are multiplexed to a pair of SMI pin interfaces on the RZ/N1 device. U-Boot assumes that the board setup code sets the SMI pin multiplexing such that GMAC1 uses SMI1, and GMAC2 uses SMI2. U-Boot provides a pair of ‘virtual’ pins to setup the multiplexing, these ‘virtual’ pins are configured using the RZ/N1 PinMux App developer tool (see the “MDIO Mux Settings” part of the tool).

It is important to note that U-Boot does not contain a driver for the 5-port switch, only helper functions to set it up. For this reason, GMAC2 is in control of SMI2, rather than the switch.

PHY Interface

The RGMII/RMII Converters and 5-port switch need to know about the link speed (1000Mbps, 100Mbps or 10Mbps) and duplex (full or half) used by the PHY. The RGMII/RMII Converters also need to know the PHY interface type (RGMII, RMII or MII) and details of the reference clock.

Using the Renesas RZ/N1-DB board code as an example, you need to:

- Modify the existing calls to `rzn1_rgmii_rmii_conv_setup()` to specify the PHY interface type. If using RMII, you must also specify whether RZ/N1 is providing the reference clock or not.
- Set GPIO[60] and GPIO[61] pin multiplexing according to the PHY interface type used for the reference clock.
- Modify `phy_adjust_link_notifier()` so that the link speed and duplex is updated by calling `rzn1_rgmii_rmii_conv_speed()` for all RGMII/RMII Converters that are in use, and `rzn1_switch_setup_port_speed()` for all ports that are connected to the 5-port switch.
- Modify the existing call to `designware_initialize()` to initialise the Synopsys GMAC1, and `designware_initialize_fixed_link()` to initialise the Synopsys GMAC2, to specify the PHY interface types.

Note: The switch port numbering runs in an opposite direction to the RGMII/RMII Converter numbering, as per the hardware.

Synopsys GMAC	RGMII/RMII Converter	5-port Switch port number
1	1	N/A
2	2	3 (also known as D)
	3	2 (also known as C)
	4	1 (also known as B)
	5	0 (also known as A)

5.9.3 Commands

There are several network commands that can be enabled. All are under the following menu:

```
Command line interface  --->
  Network commands  --->
```

For example, the tftpboot command is enabled with:

```
[*] bootp, tftpboot
```

The ping command to send an ICMP ECHO_REQUEST is enabled with:

```
[*] ping
```

The PHY MII utility commands are enabled with:

```
[*] mii
```

5.9.4 Use

The following shows an example of setting the MAC addresses, setting static IP addresses, and then downloading a file from a TFTP server. Note that `tftp` is short for `tftpboot` and is detected by U-Boot as a unique string.

```
setenv ethaddr  74:90:50:02:00:FD
setenv eth1addr 74:90:50:02:00:FE
setenv ipaddr   192.168.1.31
setenv netmask  255.255.255.0
setenv serverip 192.168.1.30
setenv gatewayip ${serverip}
ping ${serverip}
tftp 0x8ffe0000/rzn1/rzn1d400-db.dtb
tftp 0x80008000 /rzn1/uImage
bootm 0x80008000 - 0x8ffe0000
```

By default, U-Boot uses the first available network interface. If that interface is down. It will try the next one. You can set the default network interface to GMAC2, for example, by setting:

```
setenv ethact dwmac.44002000
```

5.10 SDHCI Driver

The SD card specification is an extension of MMC.

5.10.1 Configuration

The driver is enabled using:

```
Device Drivers  --->
  [ ] Support block devices
  MMC Host controller Support  --->
    [*] MMC/SD/SDIO card support
    -*- Enable MMC controllers using Driver Model
    [ ]   Support MMC controller operations using Driver Model
    [*] Arasan Generic SDHCI controller support
    [*] Secure Digital Host Controller Interface support
    [*]   Support SDHCI SDMA
```

The driver is initialised using DT. The driver additionally requires the following settings:

```
#define SDHC_CLK_MHZ                50
#define CONFIG_SDHCI_ARASAN_QUIRKS   SDHCI_QUIRK_WAIT_SEND_CMD
#define CONFIG_GENERIC_MMC
```

The SDHC_CLK_MHZ definition is also used to program the PLL clock divider for the SDHCI block.

Additionally, to access FAT partitions you need:

```
#define CONFIG_DOS_PARTITION
#define CONFIG_FS_FAT_MAX_CLUSTSIZE (8 * 1024)
```

The latter option reduces the memory requirements for U-Boot, at the expense of supporting SD cards formatted with a larger File Allocation Table.

5.10.2 Commands

The MMC commands are enabled with:

```
Command line interface  --->
  Device access commands  --->
    [*] mmc
```

The FAT commands are enabled with:

```
Command line interface  --->
  Filesystem commands  --->
    [*] FAT command support
```

5.10.3 Use

Assuming use with a SD card that has a single FAT partition, you can list files with:

```
mmc dev
fatls mmc 0
```

Load a file to address 0x80008000:

```
fatload mmc 0 80008000 <filename>
```

5.11 USB Host Driver

5.11.1 Restrictions

The USB Host Controller hardware consists of standard EHCI/OHCI controller with a PCI Bridge to the system bus. The PCI Bridge only allows two PCI-to-AHB windows to define how PCI addresses generated by the Controller are mapped to system bus (AHB) addresses. One window is configurable up to 2GB, the other is fixed at 256MB. The start address of the window must be aligned to the size of the window.

On RZ/N1D devices there are three memory regions that the Controller may need to write to, SRAM at address 0x04000000, SRAM at address 0x20000000, and DDR at address 0x80000000. In order to support all three regions, the PCI-to-AHB windows have been setup as 2GB at address 0x0, 256MB at address 0x80000000. This means U-Boot USB commands are limited to accessing the first 256MB of DDR.

5.11.2 Configuration

The driver is enabled using:

```
Device Drivers  --->
  [ ] Support block devices
  [*] USB support  --->
    [*]   USB Mass Storage support
```

The driver does not use DT. The driver requires the following settings:

```
#define CONFIG_USB_EHCI
#define CONFIG_USB_EHCI_RMOBILE
#define CONFIG_SYS_USB_EHCI_BOARD_INIT
#define CONFIG_USB_MAX_CONTROLLER_COUNT     1
#define CONFIG_SYS_USB_OHCI_MAX_ROOT_PORTS  1
```

Additionally, to access FAT partitions you need:

```
#define CONFIG_DOS_PARTITION
#define CONFIG_FS_FAT_MAX_CLUSTSIZE (8 * 1024)
```

The latter option reduces the memory requirements for U-Boot, at the expense of supporting USB mass storage devices formatted with a larger File Allocation Table.

5.11.3 Porting to different boards

The board must implement the board_usb_init() function, and this just calls rzn1_usb_init(). However, if your board uses USB in 2 x USB Host mode, your board must call the rzn1_uses_usb_func(false) function prior to use of USB. A good place for this is in board_late_init().

5.11.4 Commands

The USB commands are enabled with:

```
Command line interface  --->
  Device access commands  --->
    [*] usb
```

The FAT commands are enabled with:

```
Command line interface  --->
  Filesystem commands  --->
    [*] FAT command support
```

5.11.5 Use

Assuming use with a USB stick that has a single FAT partition, you can list files with:

```
usb start
fatls usb 0
```

Load a file to address 0x80008000:

```
fatload usb 0 80008000 <filename>
```

5.12 GPIO Driver

5.12.1 Configuration

The driver is enabled using:

```
Device Drivers  --->
  GPIO Support  --->
    -*- Enable Driver Model for GPIO drivers
    -*-   DWAPB GPIO driver
```

The driver is initialised using DT.

5.13 I2C Driver

5.13.1 Configuration

The driver is enabled using:

```
Device Drivers  --->
  I2C Support  --->
    -*- Enable Driver Model for I2C drivers
    -*- Designware I2C Controller
```

The driver is initialised using DT.

The driver additionally requires the following settings:

```
#define IC_CLK                          83
```

The IC_CLK definition is also used to program the PLL clock divider for the I2C block.

5.14 Starting Cortex-M3

On RZ/N1D and RZ/N1S devices, U-Boot includes support for starting the Cortex-M3 core. The address space that the CM3 sees is different to that with the CA7 sees in that the SRAM located at 0x04000000 is aliased to address 0x0 for the CM3. The CM3 requires its Interrupt Vector Table to be located at its address 0, therefore load the CM3 binary image to CA7 address 0x04000000. Once done, you can run the following command to start the CM3:

```
rzn1_start_cm3
```

Note: The CM3 code and data usually reside in the same SRAM blocks as are used for U-Boot and SPL. It is therefore very easy for the CM3 code to affect U-Boot and/or SPL while it is still active, causing the Cortex-A7 to crash. Care must be taken to make sure the address range of the U-Boot and SPL are kept clear, at least until the Cortex-A7 has been booted to an operating system in SDRAM.

6 U-BOOT/SPL

6.1 Introduction

U-Boot/SPL is a cut-down version of U-Boot that normally loads an image into RAM and executes it. It can read environment variables from flash memory and use this to modify arguments or data that is passed to the Linux kernel. U-Boot/SPL uses considerably less RAM than U-Boot and so is better suited to fast boot that is required for production systems.

The RZ/N1 version of U-Boot/SPL has additional features that are useful for the RZ/N1 devices. The following table provides an overview of the features supported by U-Boot and U-Boot/SPL.

U-Boot/SPL is not available for the RZ/N1L device.

Feature	U-Boot	U-Boot/SPL
Data cache enabled	✓	✓
DDR Controller setup	✓	✓
QSPI/NAND Controller read using fastest HW settings	✓	✓
QSPI/NAND Controller write	✓	×
USB mass storage device read and write	✓	×
SD card read and write	✓	×
USB DFU	✓	×
Ethernet	✓	×
Parsing and modifying Device Tree Binary (dtb) files	✓	✓
Relocating dtb to avoid Linux kernel decompression clash	✓	×
Loading multiple images	✓	✓ RZ/N1 specific
Starting the Cortex M3 core	✓	✓ RZ/N1 specific
Loading Renesas SPKG images	×	✓ RZ/N1 specific
Loading Renesas RPKG images	×	✓ RZ/N1 specific

Table 2: U-Boot/SPL Feature Comparison

6.2 Configuration

U-Boot/SPL automatically loads the next image from NAND Flash or QSPI and starts executing it. When you build U-Boot, it builds both the full command line driven version of U-Boot and U-Boot/SPL. The U-Boot/SPL Elf image that is created is `spl/u-boot-spl`.

6.3 Memory footprint

U-Boot/SPL uses the following definitions to locate the code:

```
#define CONFIG_SPL_MAX_FOOTPRINT     (57 * 1024)
#define CONFIG_SYS_SPL_MALLOC_SIZE   (38 * 1024)
#define RZN1_SPL_STACK_SIZE          (3 * 1024)
#define CONFIG_SPL_TEXT_BASE         0x040e0000
```

The addresses currently specified in the configuration files have been calculated to allow other packages to be loaded at 0x04000000, so long as they are not too big and overlap SPL's location.

Note that the current footprint corresponds to a SPL that has all the features compiled in. Features that could be disabled to reduce the footprint are:

- NAND.
  If you only use QSPI and don't need NAND support, you can disable NAND loading. This reduces the size of the heap required and code size.
- Environment support.
  The U-Boot environment is loaded by SPL to allow setting the MAC addresses to the MACs before starting Linux. If reading the MAC addresses from elsewhere, e.g. an EEPROM, this can be removed.
- I2C
  I2C may be mandatory on some boards to configure the system.

6.4 Payload location

The traditional way that SPL operates is to load a payload (i.e. a uImage) from a fixed offset into NAND or QSPI. Renesas has modified U-Boot/SPL so that it can alternatively load multiple images detailed in another type of payload (a Package Table, see below for details). Both modes of operation use the same configuration to describe the location of the payload to be loaded and executed.

If the following symbol is defined, U-Boot/SPL will compile in the capability to load from QSPI:

```
#define CONFIG_SPL_SPI_LOAD
```

A normal U-Boot/SPL build will load a U-Boot image, i.e. one created with the mkimage utility, from the media. The uImage consists of a header followed by a payload and the header contains information about the size and load address of the payload.

If loading from SPI Flash, a normal U-Boot/SPL build loads the uImage from the following offset:

```
#define CONFIG_SYS_SPI_U_BOOT_OFFS      0x20000
```

Note: U-Boot/SPL sets the SPI clock rate to that defined by CONFIG_SF_DEFAULT_SPEED.

If loading from NAND Flash, you will require this symbol:

```
#define CONFIG_SPL_NAND_LOAD
```

A normal U-Boot/SPL build loads the uImage from the following offset:

```
#define CONFIG_SYS_NAND_U_BOOT_OFFS      0x100000
```

6.5 Multiple Images and CM3 Payloads

The RZ/N1 version of U-Boot/SPL can load multiple images from NAND/QSPI, and it can start the Cortex M3 processor. The code for this is in **common/spl/spl_multi_image.c**. The ability to start the Cortex M3 processor depends on support for loading multiple-images. This is enabled with the following:

```
CONFIG_SPL_MULTIIMAGE=y
```

When this is enabled, instead of loading a uImage, U-Boot/SPL will attempt to read a *Package Table (PKGT)*. U-Boot/SPL will load a PKGT from the offset specified in the configuration. The PKGT describes a list of binary blobs to be loaded from either QSPI or NAND, and what core to start them on. U-Boot/SPL can start all CPU cores independently and is designed so that a U-Boot/SPL binary can be used in all configurations of a particular board. A single U-Boot/SPL binary can:

- Boot from QSPI or NAND without recompiling.
- Can load from QSPI or NAND regardless of boot source.
- Can start all the cores independently, in normal, NONSEC and HYP mode.
- Provide redundancy with a way to load a backup image for each core if the main code package fails to load.

6.6 Package Table (PKGT)

The Package Table is generated by the **pkgt_add_entry** tool distributed with the U-Boot source code. The table describes the details of packages to load and their attributes:

- The package media source:
  - **QSPI**: Package is loaded from QSPI, regardless of boot source.
  - **NAND**: Package is loaded from NAND, regardless of boot source
  - **"same"**: Package offset is in the boot flash.
- The image container format:
  - **uImage**: Default U-Boot image.
  - **RPKG**: Raw package (contains a header and payload).
  - **SPKG** package. Encapsulated RPKG format, also used by the RZ/N1 BootROM code.
- The processor this package is associated with:
  - Cortex **A7 #0**: the main processor core
  - Cortex **A7 #1**: This core can be started independently of Core #0
  - Cortex **M3**: Has to be loaded at 0x04000000.
- The package type:
  - **Code**: package contains code to be run, for specified core.
  - **DTB**: Device Tree Blob: For Linux or VxWorks, this device tree will be augmented with memory size allocation, Ethernet MACs etc before starting the code.
  - **Data**: Passive data. See Attributes for special cases.
- Package Attribute flags:
  - **BACKUP**: If the primary code package for a core has not been loaded successfully (for example due to bit errors on NAND) a Package marked as BACKUP will be loaded instead, it will otherwise be ignored by SPL.
    This allows a simple 'fall back' mechanism; write two Linux kernels in NAND, mark one as 'primary' and one as 'backup' and when the NAND fails, SPL will fall back to the second one.
  - **NOCRC**: Disable CRC verification on a SPKG. This is not recommended, but could be useful for very large packages that have their own integrity check.
  - **ALT**: Marks a code package as an alternative code. SPL can be configured to start an 'alternative' code when the key 'u' is pressed just at boot time. This allows the user to start U-Boot for example, instead of booting Linux.
    For this function to work, just mark the U-Boot location as 'alt' in the table, and it will be loaded and run when 'u' is pressed. This is useful to load alternative code without modifying (i.e. erasing and writing to) the package table.
  - **INITRAMFS**: If a DATA package is marked as such, its position and size will be added to the Device Tree blob as an attribute before starting Linux. This allows passing a ramdisk to the Linux kernel.
  - **NONSEC**: Switch to Non-Secure mode before starting the code package.
  - **HYP**: Switch to Hypervisor, NONSEC must also be specified. The Cortex A7 has a hypervisor mode so can start a microkernel or a hypervisor kernel, or even use kvm in Linux.
- The offset into the media (NAND or QSPI) where the image resides.

Whilst not detailed above, it is possible to load an SPKG image that has a uImage stored within it. Since a uImage has a 64-byte header before the code entry point, you simply need to specify an execution offset of 64 bytes when creating the SPKG. See the spkg_utility in section 6.9

6.7 Package Entry loading

The PKGT may contain its own redundant copies so SPL will first attempt to locate the first copy whose CRC is valid.

Prior to loading each image in the Package Table, U-Boot/SPL will:

- Load the image container header into a private buffer.
- Check whether it is allowed to load the image container format (see below).
- Check that the destination address (i.e. where the image will be loaded to) is valid.
- If an image is destined for the Cortex M3, U-Boot/SPL checks that the destination address is 0x04000000, i.e. the start of SRAM. From the perspective of the Cortex M3, this address is an alias of the reset vector at address 0x0.
- If everything is ok, U-Boot/SPL will then load the image to the destination address.

After this, U-Boot/SPL will move on to the next image in the Package Table.

When the entire table has been parsed, SPL will attempt to start each core in turn:

- If an image is destined for the Cortex M3, U-Boot/SPL will then enable the Cortex M3 clock so that it starts executing the code.
- If there is code explicitly designated as CA7#1, that code address is loaded in the SYSCTRL BOOTADDR register, and a soft interrupt is triggered on the second core for it to exit its parking mode.
- Code for the first CA7#0 is then run. If a Device Tree package has been loaded, it is augmented with various pieces of information:
    - The ram size, as probed by the SDRAM (of SDRAM is on the system).
    - Ethernet MACs for both MAC are loaded from the environment and added the /aliases/ethernet0 and /aliases/ethernet1 properties.
    - If a DATA package has been marked as ‘INITRAMFS’ its address and size are also added to the device tree.

6.8 pkgt_add_entry tool

Renesas have included the source code for a simple tool to create Package Tables, see u-boot/tools/rzn1/pkgt_add_entry.c

The tool allows creation and modification of a PKGT that can be flashed directly in QSPI or NAND.

```
RZ/N1: Add an entry to the U-Boot/SPL Package Table.
Use:
  tools/rzn1/pkgt_add_entry [<options>] -offset <offset> <file>
 Options for core selection:
  -ca70      [default]
  -ca71      Sets target core to Cortex A7 #1.
  -cm3       Set target core to the the Cortex M3.
 Options for package content:
  -code      Set packet kind to Code. [default]
  -dtb       Set packet kind to to Device Tree blob.
  -data      Set packet kind to to Data (or initramfs).
Options for package location:
```

```
  -same      Offset is in Boot flash [default].
  -qspi      Offset is QSPI.
  -nand      Offset is in NAND.
 Options for package kind:
  -u
  -uimage    Offset containts a (u-boot) uimage
  -rpkg      Is a Raw package.
  -spkg      Is a Renesas SPKG [default].
  -raw       Offset has no headers.
 Modifier flags:
  -backup    Mark this entry as an alternate to load in case
        The primary one has not been loaded due to errors.
  -nonsec    Switch to NONSEC mode for this code blob.
  -hyp       Switch to HYP mode for this code blob.
  -nocrc     Don't check SPKG payload CRC.
  -initramfs Data payload is a linux initramfs.
  -alt       Code is alternative payload (u-boot).
Optional options:
  -redundancy <times>  Number of times the Package Table is written, default
    is 128
 Mandatory options:
  -offset <offset>  Offset in source (hex)
  <file>    Filename of PKG Table file to be extended, or written if it doesn't
    exist.
 (Table size is 144 bytes; with redundancy of 8 it is 1152 total)
```

Example:

```
rm -f linux.pkgt
pkgt_add_entry linux.pkgt -o 0x200000 -code -nand -alt
pkgt_add_entry linux.pkgt -o 0x700000 -ca70 -code -hyp -nand
pkgt_add_entry linux.pkgt -o 0xb0000 -ca70 -dtb -qspi
```

This example creates a PKGT containing:

- Entry located at 0x200000 in NAND contains an ALTERNATIVE code SPKG (implied) for CA7#0 (implied). Typically, you can flash U-Boot package at that location and it'll get started if you press 'u' at start up.
- Entry located at 0x700000 in NAND contains a SPKG (implied) code package for the CA7#0 that will be started in HYP mode. Typically, that would be a Linux kernel.
- Entry located at 0xb0000 in the QSPI flash is a SPKG (implied) Device Tree Blob (dtb) for the CA7#0 core.

6.9 spkg_utility tool

Renesas have included the source code for a tool to create Secure Package Files, see u-boot/tools/rzn1/spkg_utility.c

This tool can create files that are compatible with the RZ/N1 BootROM loading code. The BootROM will only load this style of package.

```
tools/rzn1/spkg_utility
  [-i <filename>]       : input file
  [-o <filename>]       : output file
  [--load_address <hex constant>] : code load address
  [--execution_offset <hex constant>] : starting offset
  [--nand_ecc_enable] : Enable nand ECC
  [--nand_ecc_blksize <hex constant>] : Block size code
  [--nand_ecc_scheme <hex constant>] : ECC scheme code
  [--add_dummy_blp] : Add a passthru BLP
  [--padding <value>[K|M]] : Pass SPKG to <value> size block
```

Example:

```
spkg_utility -i u-boot.bin \
    -o u-boot.bin.spkg \
    --padding 64K --load_address 0x200a0000 \
    --nand_ecc_enable \
    --nand_ecc_blksize 1 --nand_ecc_scheme 1 --nand_bytes_per_ecc_block 7 \
    --add_dummy_blp
```

This creates an SPKG containing the u-boot bootloader. The U-Boot code will be loaded at 0x200a0000, and we also specify the NAND ECC flags needed by the ROM to be able to do error correction. See the BootROM documentation for details.

The ‘add_dummy_blp’ will insert a ‘dummy header, so the BootROM will load the code.

The tool will create this SPKG file as an exact number of 64K blocks, by padding as necessary. This allows the file to be flashed on NAND and use whole erase blocks.

6.10 Use

A typical use for U-Boot/SPL is to load a Cortex M3 image, a DTB and the Linux kernel. The BSP includes two scripts that can be used as a template for this scenario:

`${RELEASE_DIR}/u-boot/spl-gen-pkg-table.sh` generates the SPKG files, then creates a PKGT package table image.

`${RELEASE_DIR}/u-boot/spl-dfu-pkg-table.sh` can be used to program the images into QSPI.

Both scripts optionally include the Cortex M3 files. If they are not present in the locations specified in scripts, they will not be included.

7 RENESAS RZ/N1D-DB BOARD

7.1 Memory Use

The RZ/N1-DB board configuration as provided uses the following memory map.

Address	Use	Size	Related Options
0x88000000	Top of DDR		CONFIG_SYS_SDRAM_BASE + CONFIG_SYS_SDRAM_SIZE
0x80000000	DDR	256MiB	CONFIG_SYS_SDRAM_BASE
0x20100000	Top of SRAM		
0x200F0000	Reserved for use by the BootROM, e.g. when it downloads an SPKG via DFU.	64KiB	CONFIG_SYS_INIT_SP_ADDR+ CONFIG_SYS_MALLOC_F_LEN
0x200F0000	Initial heap	1KiB	CONFIG_SYS_INIT_SP_ADDR CONFIG_SYS_MALLOC_F_LEN
0x200A0000	U-Boot code image. A small stack used early on is placed at the top of this region.	320KiB	CONFIG_SYS_TEXT_BASE
0x20000000	Unused SRAM	640KiB	
0x04100000	Top of SRAM		CONFIG_SYS_SRAM_BASE + CONFIG_SYS_SRAM_SIZE
0x040FC000	MMU TLB Page Table	16KiB	
0x040FA000	Environment variables	8KiB	CONFIG_ENV_SIZE
0x040AA000	Heap	320KiB	CONFIG_SYS_MALLOC_LEN
0x040A9EF0	Board Info	sizeof(gd_t), 0x80	
0x040A9E60	Bottom of IRQ stack	144B	
0x040A8000	Bottom of stack	7KiB	
0x04000000	Unused SRAM. Note that Cortex M3 code must place its vector table at address 0x04000000.	672KiB	CONFIG_SYS_SRAM_BASE

Table 3: RZ/N1-DB Board Memory Map

7.2 Ethernet

The board is setup to use:

- GMAC1 via a Marvel 88P1512 PHY with MDIO0/MDC0, configured for RGMII interface, PHY address 8. This is only available if the board is used with the Extension Board.
- GMAC2 via Micrel KSZ8041TL PHYs on MII 4 and MII 5 interfaces, both configured for MII interface, both with MDIO1/MDC1. U-Boot can only access a single PHY per GMAC, and is setup to use MII 4 (PHY address 4), available via connector CN1.

This is shown in the diagram below.

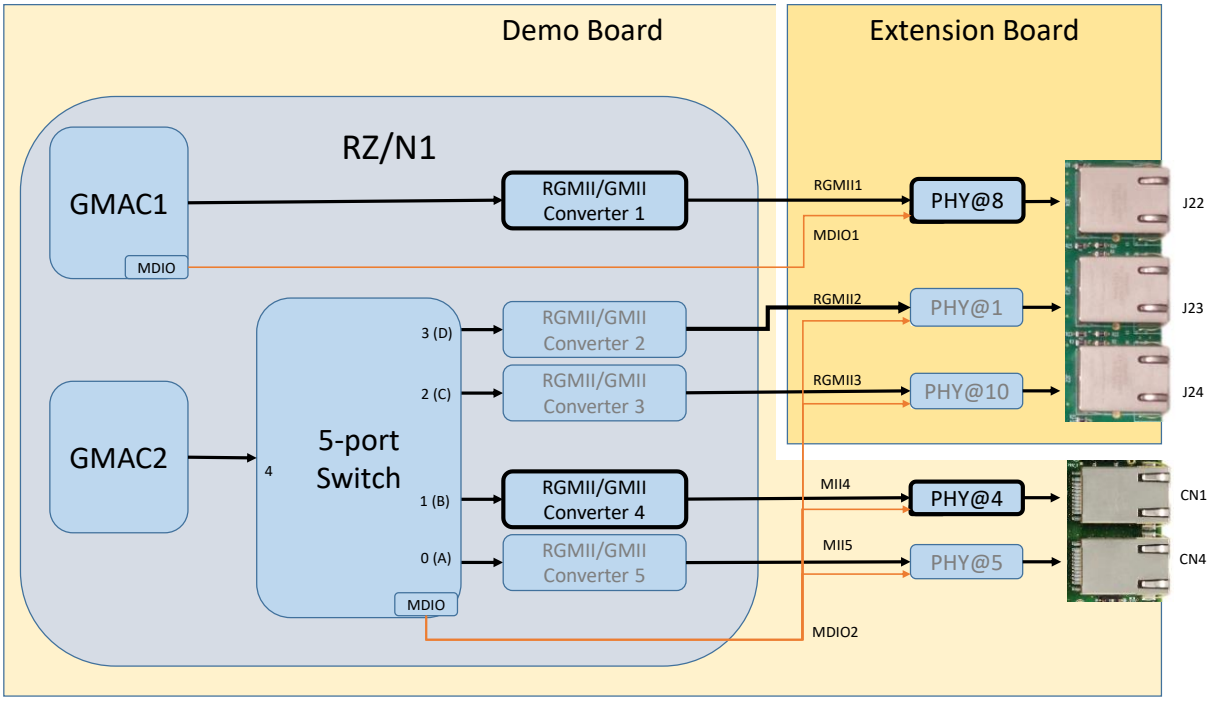

Figure 2 RZ/N1-DB Board Ethernet

8 ACRONYMS

CPU	Central Processing Unit
CA7	Cortex A7
CM3	Cortex M3
DFU	Device Firmware Upgrade (USB protocol)
DDR	Double Data Rate (memory)
GMII	Gigabit MII (Ethernet PHY interface)
MAC	Media Access Control (Ethernet)
MII	Media-Independent Interface (Ethernet PHY interface)
PKGT	Package Table, as loaded by SPL in “Multi” package mode.
QSPI	Quad SPI
PHY	Physical (interface)
RGMII	Reduced GMII (Ethernet PHY interface)
RPKG	Raw Package
SMI	Serial Management Interface (for communication to Ethernet PHYs using MDIO/MDC)
SPI	Serial Peripheral Interface
SPL	Secondary Program Loader
SPKG	Secure Package – encapsulated RPKG package, as loaded by the RZN1 ROM

9 REFERENCES

1. DFU Utils Installation:
https://community.spark.io/t/tutorial-installing-dfu-driver-on-windows-15-nov-2014/3518

10 CHANGE HISTORY

Version	Description	Date
1.0	First public release	13th Apr 2017
1.01	Added USB Host support Minor corrections	10th July 2017
1.02	DTB address moved to top of DDR. SPI Flash commands added. Improve Windows dfu-util instructions. Moved U-Boot/SPL into a separate section. Minor corrections	18th Oct 2017
1.03	Moved to external git repo Added RZ/N1L-DB board	15th Jan 2018
1.04	Removed spkg_utility --config option. Corrected memory used by the BootROM in Table 3: RZ/N1-DB Board Memory Map USB porting information added. GMII corrected to RMII. Added details about clock management.	11th Jul 2018
1.05	Changed build setup instructions to refer to exact version. U-Boot/SPL pkgt_add_entry has '-redundancy' option added.	11th Oct 2018
1.06	3.2.2: Fixed URL for Zadig	5th Jul 2019

RZ/N1 U-Boot

Renesas Electronics Corporation

R01US0296EG0106