Users Manual

RZ/N1 Linux

System-on-Chip

Target Device RZ/N1

Renesas Electronics
www.renesas.com

R01US0297EG0110
August 30, 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

LIST OF TABLES

1 OVERVIEW

1.1 Overview

This manual describes the Linux software developed by Renesas for the RZ/N1 device. The software consists of Yocto recipes, U-Boot and the Linux kernel ported to the RZ/N1 device. U-Boot information is provided in a separate U-Boot User manual, as it is also used by other operating systems.

The software supports the following boards:

- Renesas RZ/N1D-DB Board with the Extension Board.

1.2 Functionality

The software includes the following functionality:

- Yocto recipes to build
    - Root file system
    - U-Boot
    - Linux Kernel

- Linux Kernel
    - Drivers for:
        - Clock Controller
        - PinMux
        - UART
        - DDR ECC
        - QSPI
        - NAND Flash
        - Ethernet MAC
        - 5-port Switch
        - RGMII/RMII Converters
        - SDHC
        - I2C
        - USB Function
        - USB Host
        - LCD
        - CAN
        - DMAC
        - SPI
        - RTC

2 BUILD SETUP

2.1 Development Environment

The software package 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.

2.2 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
```

Note: Although untested, 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/a
rm-linux-gnueabihf/gcc-linaro-6.3.1-2017.02-i686_arm-linux-gnueabihf.ta
r.xz
```

2.3 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 libssl-dev u-boot-tools gettext bison flex lzop
```

2.4 Yocto

These instructions provide RZ/N1 specific information. For full details of setting up and using Yocto, please see the Yocto project Quick Start guide at:
http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html.

The root file system built is based on the Yocto `core-image-minimal` recipe, but has a few extra packages installed that are useful for development and testing.

1. RZ/N1 uses the Rocko (v2.4) release of Yocto, therefore after setting up prerequisites run:
```
git clone http://git.yoctoproject.org/git/poky
cd poky
git checkout -b rocko-rzn1 yocto-2.4.3
```

RZ/N1 requires patches to the Rocko release to support the VPFv4d16 floating-point coprocessor. To apply them, run:
```
git am ${RELEASE_DIR}/yocto/rocko/0001-ARM-Add-Cortex-A7-vfpv4-d16*.patch
cd ..
```

2. Clone the OpenEmbedded recipes:
```
git clone -b rocko git://git.openembedded.org/meta-openembedded
```

3. Support for a read-only root file system, with an overlay read-write file system, is provided using the meta-readonly-rootfs-overlay layer. Clone the layer:
```
git clone https://github.com/cmhe/meta-readonly-rootfs-overlay.git
cd meta-readonly-rootfs-overlay
git checkout -b rzn1 2a426495fe77330058bc0d6ef98e914649e7b415
cd ..
```

4. Support for RZ/N1 is provided using a meta-rzn1 layer. Clone the layer:
```
git clone -b rocko-v4.19 https://github.com/renesas-rz/meta-rzn1.git
```

5. To define the OpenEmbedded build environment settings needed to complete the build, run:
```
cd poky
source oe-init-build-env
```

6. Copy ${RELEASE_DIR}/yocto/bblayers.conf to build/conf/bblayers.conf. This file specifies what layers are used for the build.

7. Copy ${RELEASE_DIR}/yocto/local.conf to build/conf/local.conf. This file specifies what you want to build.
   a. This file specifies 24 CPU cores for number of threads and parallel make. You may wish to reduce this for typical PCs, see the definition of BB_NUMBER_THREADS and PARALLEL_MAKE.
   b. If you want to add or remove packages to/from the root file system, edit the definition of IMAGE_INSTALL_append.
   c. Optional: If you wish to build a kernel with initramfs, add:
```
INITRAMFS_IMAGE = "core-image-minimal"
INITRAMFS_IMAGE_BUNDLE = "1"
IMAGE_FSTYPES = "${INITRAMFS_FSTYPES}"
```

8. Finally, you can build the root file system that you want to use, and the kernel, using:
```
bitbake core-image-minimal
```

If you just want to build the kernel, use:

```
bitbake linux-rzn1
```

To build U-Boot, use:

```
bitbake u-boot-rzn1
```

The generated files are in the tmp/deploy/images/rzn1d400-db directory.

Note: The tarball root file systems do not have a link added from init to sbin/init. If you intend to unpack the tarball and use this when building the kernel with an initramfs for kernel development outside of Yocto, make sure you add this soft link.

2.5 Linux Kernel only

These instructions are for building the Linux kernel directly, without using Yocto. It assumes you already have a root file system available, for example built with BuildRoot or Yocto.

There are stable versions of Linux v4.19, e.g. v4.19.1, v4.19.2, etc, and we recommend that you use the latest stable version. In addition, you may want to use the Real-Time Linux. RT-Linux is available for most stable versions, however RT releases may lag the release of stable versions.

These instructions use the stable RT-Linux git repository.

2.5.1 Setup

The following steps on your host PC will download the RZ/N1 Linux kernel source code, including all commit history. Note that it will result in a large download.

Clone the Linux stable RT repository:

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git
cd linux-stable-rt
```

Checkout a branch based on the latest v4.19 stable RT version:

```
git checkout -b rzn1 origin/v4.19-rt
```

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

```
BSP_VERSION=v1.6.0
```

Fetch the RZ/N1 branch and merge it in:

```
git remote add renesas-rz https://github.com/renesas-rz/rzn1_linux.git
git fetch --tags renesas-rz
git merge rzn1-$BSP_VERSION
```

There may be merge conflicts that you have to resolve manually, see git documentation.

2.5.2 Build

The patches provided already enable the relevant kernel modules. To setup the default configuration for RZ/N1 systems, run:

```
make rzn1_defconfig
```

If you wish to make any changes to the configuration, e.g. enable RT-Linux, run:

```
make menuconfig
```

To enable RT-Linux, select:

```
  General Setup  --->
    Preemption Model --->
      (X) Fully Preemptible Kernel (RT)
```

To build the kernel, run:

```
make LOADADDR=80008000 uImage
```

Build the Device Tree blob (dtb), e.g. for the Renesas RZ/N1-DB board, run:

```
make rzn1d400-db.dtb
```

Once complete, the kernel uImage/zImage is stored in arch/arm/boot and the dtb is stored in arch/arm/boot/dts.

3 LINUX KERNEL

The kernel port is based on Linux v4.19 and consists of machine start up code, located in arch/arm/mach-rzn1, device tree files, driver for the clock controller, and pin multiplexing.

Important Note: Throughout the software, the indexes used to enumerate hardware start at 0, whereas the documentation for the RZ/N1 device starts at 1.

3.1 Device Tree

When using a Device Tree Blob (dtb), the dtb provides platform identification, runtime configuration and device population. The dtb is built from the Device Tree Source (dts) file. All ARM dts files are located in the arch/arm/boot/dts directory.

Note: The IRQ numbers used in the dts are offset from those shown in the RZ/N1 User's Manual. This is because, as per other ARM dts files, the interrupt numbers start after the 16 SGI interrupts.

3.2 SMP

By default, the kernel is configured to use both cores in SMP mode.

Note: Normally, Linux uses individual power domain control for each CPU to bring additional CPUs online. The RZ/N1 device does not have power domain control for individual cores, hence the need for a holding pen. The holding pen is implemented in a way that allows booting in Secure and Non-Secure modes.

When booting in Secure mode, Linux simply requires that the BootROM has started the 2nd core, has executed WFI or WFE, and on exit will jump to the address in the System Controller's BOOTADDR register.

When booting in Non-Secure mode, the kernel requires the use of a secondary holding pen because the BOOTADDR register can only be accessed in Secure mode. RZ/N1 Linux requires the Device Tree to contain a property called "`rzn1,bootaddr`" that specifies the address of this holding pen; this property must be in the "`/chosen`" node. The RZ/N1 version of U-Boot automatically added this property when built with CONFIG_ARMV7_NONSEC.

To use only one core, add "maxcpus=1" to the kernel command line in the device tree.

From a console, you can make an application run on a specific CPU by using the taskset utility. For example, the following command starts 'top' running on the 2nd CPU:

```
taskset 2 top
```

3.2.1 Interrupts

By default, the kernel handles all interrupts on the 1st CPU. Individual interrupts can be moved to the 2nd CPU using simple command such as:

```
echo 2 > /proc/irq/352/smp_affinity
```

Note: Interrupts have only been tested on the 1st CPU by Renesas, as this is standard functionality for ARM processors.

3.3 Booting a zImage

The following table details the registers that are required to be setup by a boot loader. These are common for all ARM v7 devices and provided here for information purposes only.

Register	Function with DT	Function prior to DT
R0	Always 0	Always 0
R1	Not used	Machine ID (arch/arm/tools/mach-types)
R2	Pointer to dtb	Pointer to ATAGs
PC	Start of kernel image	Start of kernel image

Table 1 – Boot Registers

It is important to note that Renesas has implemented a complete system in U-Boot, SPL and the Linux kernel that requires the use of device tree blob and does not support the old boot method (ATAGS). In particular, SMP and NONSEC specifically require a device tree blob.

Since the kernel requires space to perform decompression, care must be taken to ensure that the location of the dtb, and any other data such as an initramfs, are suitably placed. The most common approach is to place the dtb at the start of DDR, and the zImage at start of DDR + 0x8000. However, note that the address for a valid dtb must be 64-bit aligned and cannot be zero as this indicates that a dtb is not present.

Further information about Device Trees and how they are used can be found in the kernel documentation, for Device Tree bindings, please see Documentation/devicetree/usage-model.txt.

3.4 Clock Controller

This driver supports enabling and disabling clocks, and obtaining the clock rates. Note that this driver does not modify any of the peripheral clock dividers that are set by writing to registers in the RZ/N1's System Controller (SYSCTRL) block. It is assumed that software that loads the Linux kernel also sets these clock dividers to appropriate values.

The driver is always enabled for RZ/N1.

The complete clock tree description used by Linux and the clock driver group is in the file:

```
./arch/arm/boot/dts/rzn1-clocks.dtsi
```

That file was generated from device design files, and is meant to be included by any device tree description files for specific boards.

It is possible to dump the state of the clock tree in Linux, including which clocks are used, and at what frequency by using these commands:

```
mount -t debugfs / /sys/kernel/debug
cd /sys/kernel/debug/
cat /sys/kernel/debug/clk/clk_summary
```

It is also possible to change some of the clock dividers at runtime by using a clock driver system special file located at:

```
/sys/kernel/rzn1/clk_set_rate
```

This file contains a short message explaining its usage:

```
Usage: echo <clock name> <rate in hz> >clk_set_rate
```

For example, when using the 'performance' CPUFreq governor, you can change the frequency of the CPU clock by using this command:

```
echo div_ca7 250000000 >/sys/kernel/rzn1/clk_set_rate
```

Note: You can obviously hang the system very easily by doing the wrong thing with this file, there are no checks of parameter sanity.

3.5 Dynamic Frequency Control

This driver supports changing the CPU clock divider.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
CPU Power Management  --->
    CPU Frequency scaling  --->
        <*>   Generic DT based cpufreq driver
```

The default CPUFreq governor for RZ/N1 is set to ‘ondemand’, this scales the CPU frequency dynamically according to current load. It jumps to the highest frequency and then backs off as the idle time increases. There are several other governors that can be enabled, for example ‘performance’ sets the CPU frequency to the maximum frequency that is supported.

3.5.1 Userspace control

Change the governor to ‘performance’

```
echo performance > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

Change the governor to ‘ondemand’

```
echo ondemand > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

Note: When using the ‘performance’ governor, you can program a fixed CPU frequency using the Clock Controller; see chapter 3.4 for an example.

3.6 UART

This driver supports serial ports using the Synopsys DesignWare UARTs.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    Character devices  --->
      Serial drivers  --->
        [*] 8250/16550 and compatible serial support
        [*]   Console on 8250/16550 and compatible serial port
        [*] Support for Synopsys DesignWare 8250 quirks
```

For Device Tree bindings, see Documentation/devicetree/bindings/serial/snps-dw-apb-uart.txt.

An additional DT binding has been added to specify the Component Parameter Register (CPR) value. The CPR register is not present on the RZ/N1 devices but provides the driver with information about the FIFO depth, Auto Flow Control and DMA capabilities.

The driver has been modified to support DMA specifically for RZ/N1 where the UART is acting as the flow controller. The driver uses the ‘character timeout’ interrupt to determine when to push data up the stack to the user or another driver. This interrupt is only generated when there is at least one byte in the receive FIFO. Therefore, care needs to be taken when setting the receive threshold, the DMA source burst size, along with the DMA destination burst size and memory width setting. Once the receive threshold is met, a DMA request to read data from the receive FIFO is made. To ensure the FIFO is not entirely emptied, the DMA source burst size is set to a quarter of the FIFO depth. However, when processing a ‘character timeout’ interrupt, we must ensure the DMA Controller has written to memory all data read from the receive FIFO. When pausing DMA, the DMAC can only write data to memory that is a multiple of the memory width setting, anything less than this is discarded. Therefore, the UART driver also limits the memory width setting when reading data from the receive FIFO.

3.7 I2C

This driver supports I2C using Synopsys DesignWare I2C.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    I2C support  --->
      [*] I2C support
           I2C Hardware Bus support  --->
              [*] Synopsys DesignWare Platform
```

For Device Tree bindings, see Documentation/devicetree/bindings/i2c/i2c-designware.txt.

For information on using i2c, see Documentation/i2c/dev-interface.

For basic read/write via i2c, we recommend using the i2c-utils package.

3.8 SPI

This driver supports SPI Master mode using the RZ/N1 SPI Controller that is based on the Synopsys DesignWare SSI.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] SPI support  --->
      [*]   DesignWare SPI controller core support
      [*]     Memory-mapped io interface driver for DW SPI core
```

The driver has been modified to support DMA and specifically for RZ/N1 where the SPI Controller is acting as the flow controller. The driver has also been modified to add support for the modified CS line toggling mode of operation.

For Device Tree bindings, see Documentation/devicetree/bindings/spi/snps,dw-apb-ssi.txt in the provided Linux source code.

3.9 QSPI Serial Flash

This driver supports the Cadence QSPI Controller.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Memory Technology Device (MTD) support  --->
        [*]   SPI-NOR device support  --->
            [*]   RZ/N1 Cadence Quad SPI controller
```

For Device Tree bindings, see Documentation/devicetree/bindings/mtd/renesas,rzn1-qspi.txt in the provided Linux source code.

3.10 SDHC

This driver supports SD Host Controller using Arasan SD3.0/ SDIO3.0/ eMMC4.51 Host Controller.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] MMC/SD/SDIO card support  --->
      [*]   Secure Digital Host Controller Interface support
      [*]     SDHCI platform and OF driver helper
      [*]       SDHCI OF support for the Arasan SDHCI controllers
```

For Device Tree bindings, see Documentation/devicetree/bindings/mmc/arasan,sdhci.txt.

3.11 NAND Flash

This driver supports the Cadence NAND Flash Controller.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Memory Technology Device (MTD) support  --->
        [*]   Raw/Parallel NAND Device Support --->
            [*]   Enable Evatronix NANDFLASH-CTRL driver
```

The driver is based on upstream patches to add a driver for the Evatronix NAND Flash controller. Note: Evatronix were purchased by Cadence, but otherwise the IP is identical.

For Device Tree bindings, see Documentation/devicetree/bindings/mtd/evatronix-nand.txt in the provided Linux source code.

3.12 USB Host

This driver supports USB Host using the Renesas USB TYPE-H2F2 hardware. The Host controller sits behind a PCI bridge, so Linux sees the USB Controller as a PCI card. Since the Host Controller uses the standard EHCI/OCHI register set, no driver is required other than the PCI bridge.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Bus support  --->
    [*] PCI support
        PCI controller drivers --->
            [*] Renesas R-Car Gen2 Internal PCI controller
Device Drivers  --->
    [*] USB support  --->
        [*]   Support for Host-side USB
        [*]     EHCI HCD (USB 2.0) support
        [*]     Generic EHCI driver for a platform device
        [*]     OHCI HCD (USB 1.1) support
        [*]       OHCI support for PCI-bus USB controllers
        [*]       Generic OHCI driver for a platform device
```

The driver has been specifically written for the RZ/N1 device.

For Device Tree bindings, see Documentation/devicetree/bindings/pci/pci-rcar-gen2.txt in the provided Linux source code.

3.13 USB Function

This driver supports USB Gadgets using the Renesas USB TYPE-H2F2 hardware.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] USB support  --->
        [*]   USB Gadget Support  --->
              USB Peripheral Controller  --->
              [*] Renesas USB Function Peripheral Block (RZ/N1)
        < Select appropriate USB Gadget Driver >
```

The driver supports DMA, all the endpoint modes, and internal SRAM partitioning profiles via a set of device tree properties.

For Device Tree bindings, see Documentation/devicetree/bindings/usb/renesas,usbf.txt in the provided Linux source code.

3.14 Ethernet MAC

This driver supports the Synopsys Gb Ethernet MAC.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Network device support  --->
      [*]   Ethernet driver support  --->
        [*]   STMicroelectronics devices
        <*>     STMicroelectronics 10/100/1000 Ethernet driver
        <*>       STMMAC Platform bus support
        <*>         Generic driver for DWMAC
Device Drivers  --->
    -*-   PHY Device support and infrastructure  --->
        < Select appropriate PHYs >
```

For Device Tree bindings, see Documentation/devicetree/bindings/net/stmmac.txt in the provided Linux source code.

3.15 RGMII/RMII Convertors

This driver supports the RGMII/RMII Convertors found on the RZ/N1 device.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Network device support  --->
      [*]   Ethernet driver support  --->
        [*]   Renesas devices
        <*>     Renesas RZ/N1 RGMII/GMII Convertor
```

The driver has been specifically written for the RZ/N1 device. The RGMII/RMII Convertors driver sets up the PHY interface type (RGMII, RMII or MII) accordingly and handles changes to the PHY link speed and duplex.

3.16 5-Port Switch

This driver supports the MoreThanIP 5-Port Switch.

Note: this driver is only used in the “No CM3” mode, see section 4.1.3.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Network device support  --->
      [*]   Ethernet driver support  --->
        [*]   MoreThanIP devices
        <*>     Embedded MoreThanIP 5-port Ethernet switch
```

The driver has been specifically written for the RZ/N1 device. It implements an MDIO bus driver, and registers Ethernet network devices for each downstream port that is connected to a PHY. If the Device Tree node for the Switch does not contain PHY addresses for all the downstream ports, the driver will automatically probe and add PHY addresses for any PHY that responds. This process can take a long time and so can affect the boot time. Therefore, we recommend that if a downstream port is unused on a board, the Device Tree contains a PHY address that is not in use.

The 5-port Switch driver handles changes to the PHY link speed and duplex. The driver supports changing PHY settings using *mii-info* and *ethtool*.

3.17 CAN

This driver supports the CAN Controller.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
[*] Networking support  --->
  [*]   CAN bus subsystem support  --->
    [*]   Raw CAN Protocol (raw access with CAN-ID filtering)
    [*]   Broadcast Manager CAN Protocol (with content filtering)
    [*]   CAN Gateway/Router (with netlink configuration)
          CAN Device Drivers  --->
            [*] Platform CAN drivers with Netlink support
             [*]   CAN bit-timing calculation
             [*]   Philips/NXP SJA1000 devices  --->
                [*]   Generic Platform Bus based SJA1000 driver
```

The driver has been modified to support the RZ/N1. This IP does not have a Clock Divider Register (CDR), and does not echo transmitted messages. Additionally, the driver has been modified to get a clock for the IP and use this to get the clock rate.

The driver only supports the “CAN Reduce” functionality, it does not support transmitting periodic “Sync frame”, or triggers connected to the Motor Control.

For Device Tree bindings, see Documentation/devicetree/bindings/net/can/sja1000.txt in the provided Linux source code.

For information on how to configure the CAN driver, see section 6.5 “The CAN network device driver interface” of Documentation/networking/can.txt.

3.18 Timers

This driver supports the RZ/N1 timers.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    Clock Source drivers  --->
        [*] Renesas RZ/N1 Timer
```

The driver has been specifically written for the RZ/N1 device.

3.19 Watchdog

This driver is capable of triggering a device reset in case the system fails to update its counter. To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Watchdog Timer Support  --->
        [*]   Renesas RZ/N1 watchdog
```

Enabling this driver will create device files `/dev/watchdogX` that can be used by for example BusyBox `watchdogd`.

To enable or kick the Watchdog:

```
echo 'v' > /dev/watchdog
```

Note: The Watchdog will only reset the device if the SysCtrl RSTEN register has the WDA7RST_EN bits set. This is not done by the Watchdog driver as the behaviour depends on use.

3.20 RTC

This driver supports the real-time clock using Renesas RTC.

This clock will be read at boot time by the Linux kernel. To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] Real Time Clock  --->
        [*]   Renesas RZN1
```

The driver has been specifically written for the RZ/N1 device.

You can read and update the hardware date using BusyBox `hwclock` command. For example, to set the current date and time:

```
date -s "2019-08-28 16:21:00"
hwclock -w
```

To read the current date and time:

```
hwclock
```

3.21 PinCtrl

This driver supports the Renesas Port Multiplexer.

The driver is always enabled for RZ/N1.

Interaction with this driver is done exclusively via descriptions in the device tree files. The PinMux App generates a device tree file that contains all the pin multiplexing, drive strength and pull up/down information used by a board.

A typical peripheral pin group can be seen as:

```
  pins_i2c1: pins_i2c1 {
            pinmux = <
                    RZN1_PINMUX(115, RZN1_FUNC_I2C) /* I2C1_SCL */
                    RZN1_PINMUX(116, RZN1_FUNC_I2C) /* I2C1_SDA */
            >;
            drive-strength = <12>;
  };
```

This pin group describes the function of the two pins 115 and 116 which correspond to the hardware pins PL_GPIO[115] and PL_GPIO[116] – these two pins will be configured as function I2C. This corresponds to Level 2 hardware function 50.

Note: These pin mux specifications do not change the pullup/pulldown settings on any pins if any have been set by the bootloader, unless the RZN1_MUX_PUP and RZN1_MUX_PDOWN macros are used instead. Macros are defined to change the drive strength if necessary, see file include/dt-bindings/pinctrl/pinctrl-rzn1.h for details.

3.22 DMAC

This driver supports peripheral to memory and memory to peripheral DMA using the Synopsys DesignWare DMAC.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] DMA Engine support  --->
      [*]   Synopsys DesignWare AHB DMA platform driver
```

The driver has been modified for the RZ/N1 to allow the SYSCTRL DMA request multiplexing to be specified via Device Tree properties. Additionally, the driver has been modified to handle situations where the peripheral is acting as the flow controller.

For Device Tree bindings, see Documentation/devicetree/bindings/dma/snps-dma.txt in the provided Linux source code.

An example of the DMA properties for SPI0 is:

```
dmas =  <&dma0 8 0 0>, <&dma0 9 0 0>;
dma-names = "rx", "tx";
```

Drivers usually request DMA channels by their names, i.e. “rx” or “tx” in the example above, and then get the corresponding DMA channel information from the dmas DT property. For example, “rx” corresponds to dma0, channel 8 and dma0 is the name of the DT node that corresponds to the DMAC0 IP block. The channel numbers are fixed based in hardware, see the RZ/N1 User’s Manual DMA Controller “DMA Channel Allocation” section.

Additional examples of DMA properties are shown below.

SPI1:

```
dmas =  <&dma0 10 0 0>, <&dma0 11 0 0>;
dma-names = "rx", "tx";
```

SPI2:

```
dmas =  <&dma0 12 0 0>, <&dma0 13 0 0>;
dma-names = "rx", "tx";
```

SPI3:

```
dmas =  <&dma0 14 0 0>, <&dma0 15 0 0>;
dma-names = "rx", "tx";
```

UART3:

```
dmas =  <&dma0 0 0 0>, <&dma0 1 0 0>;
dma-names = "rx", "tx";
```

UART4:

```
dmas =  <&dma0 2 0 0>, <&dma0 3 0 0>;
dma-names = "rx", "tx";
```

UART5:

```
dmas =  <&dma0 4 0 0>, <&dma0 5 0 0>;
dma-names = "rx", "tx";
```

UART6:

```
dmas =  <&dma0 6 0 0>, <&dma0 7 0 0>;
dma-names = "rx", "tx";
```

UART7:

```
dmas =  <&dma1 4 0 0>, <&dma1 5 0 0>;
dma-names = "rx", "tx";
```

Note: UART7 uses DMAC1, whereas the others use DMAC0.

Note: These DMA properties cannot be added to the rzn1.dtsi because the RZ/N1 multiplexes the DMA handshaking signals with other IP blocks.

3.23 LCD Controller

This driver supports the Digital Blocks DB9000 LCD Controller.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    Graphics support  --->
        Frame buffer Devices  --->
            [*] DB9000 LCD framebuffer support
            [*]   DB9000 LCD auto-blink support
        [*] Backlight & LCD device support  --->
            [*]   Lowlevel Backlight controls
            [*]     Digital Block DB9000 LCDC Contrast-as-Backlight control
```

The driver is based on the out of tree ST Spear driver for the same IP block. The driver has been modified with Device Tree support and to support the RZ/N1 special ‘blinking’ mode.

For Device Tree bindings, see Documentation/devicetree/bindings//fb/db9000fb.txt in the provided Linux source code.

Note: If used as a module, you must load the "Framebuffer Console" module (fbcon) after loading the db9000fb module.

3.23.1 Pixel clock

For the pixel clock source, the LCD controller hardware uses a dedicated clock source that is controlled by the SYSCTRL clock divider (PWRCTRL_PG4_PR1DIV register. The SYSCTRL clock divider provides a clock that is 1GHz divided down by any value between 12 and 200. This ensures the achievable pixel clock is within 4% of any desired frequency range.

The driver reads the desired pixel clock from the device tree, i.e. the `clock-frequency` property in the `display-timings` node. The driver calculates the nearest pixel clock that can be achieved via the clock divider that is under or equal to the desired frequency. Therefore, it is advisable to set the `clock-frequency` property to the maximum supported by the LCD panel used.

3.23.2 Userspace control

Blank the screen

```
echo 1 > /sys/class/graphics/fb0/blank
```

Un-blank the screen

```
echo 0 > /sys/class/graphics/fb0/blank
```

Set the PWM backlight brightness, where *val* is in the range 0 to 255

```
echo val > /sys/class/backlight/backlight/brightness
```

Turn off the blinking cursor

```
echo 0 > /sys/class/graphics/fbcon/cursor_blink
```

Change the frame buffer format to RGB565 (i.e. 16 bits per pixel)

```
fbset -depth 16
```

Change the frame buffer format to packed RGB888 (i.e. 24 bits per pixel)

```
fbset -depth 24
```

Change the frame buffer format to unpacked xRGB888 (i.e. 32 bits per pixel)

```
fbset -depth 32
```

3.24 DDR Controller

This driver implements an Error Detection and Correction (EDAC) driver for the Cadence DDR Controller.

To enable the driver, make the following settings using the kernel “make menuconfig” command.

```
Device Drivers  --->
    [*] EDAC (Error Detection And Correction) reporting  --->
        [*]   Main Memory EDAC (Error Detection And Correction) reporting
        [*]     Cadence Memory Controller
```

The driver has been specifically written for the RZ/N1 device. It will detect DDR errors and report them via sysfs, e.g.:

```
cat /sys/devices/system/edac/mc/mc0/ce_count
cat /sys/devices/system/edac/mc/mc0/ue_count
```

You can inject an ECC error to check correct handling in the driver by writing a syndrome that reflects the type of error to a sysfs entry, e.g.

```
echo 0x75 > /sys/devices/system/edac/mc/mc0/inject_ctrl
```

Please see the Cadence DDR Controller User Manual section 11.4 Syndromes for details of the values that can be used. Note that there is no way to control the address that will suffer the ECC error, so do not inject un-correctable multi-bit ECC errors.

4 RZ/N1D-DB BOARD SPECIFICS

Note that throughout the software, the indexes used to enumerate hardware start at 0, whereas the documentation for the RZ/N1 device starts at 1.

4.1 Configurations

The BSP includes multiple Device Tree Source files for the RZ/N1D-DB board to support different networking setups. In all cases, Linux controls all the hardware that is not related to networking.

4.1.1 Normal Mode

Device Tree Source: rzn1d400-db.dts

Linux runs on the Cortex A7s and controls GMAC1 and the external PHY connected via RGMII1 using MDIO1.

The Cortex M3 controls the HW-RTOS GMAC and/or GMAC2, the 5-Port Switch, and the RGMII/RMII Convertors connected to the 5-Port Switch. The Cortex M3 controls the external PHYs connected to the 5-Port Switch. The HW-RTOS GMAC or GMAC2 is connected to the 5-Port Switch's upstream (in) port. See Figure 1 for details.

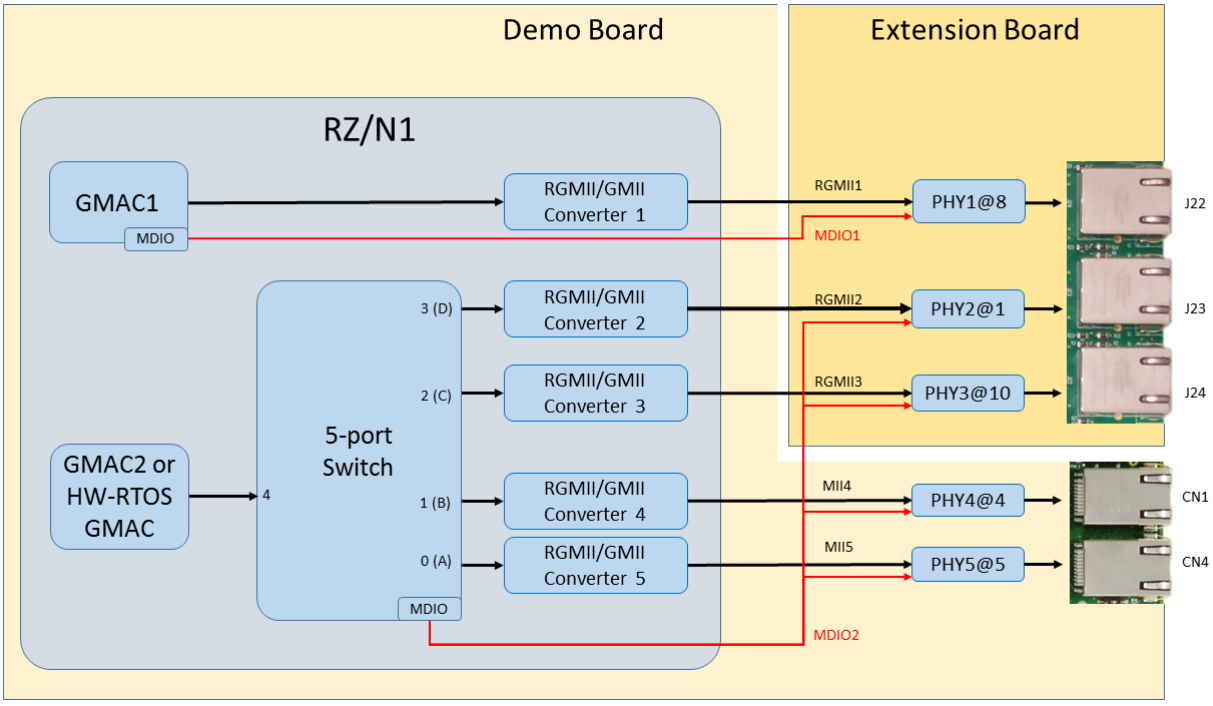

Figure 1 Normal Mode Ethernet

Ethernet is only available when using the RZ/N1D-DB board with the Extension Board.

4.1.2 Master Mode

Device Tree Source: rzn1d400-db-both-gmacs.dts

Linux runs on the Cortex A7s and controls GMAC1, GMAC2, and the external PHYs connected via RGMII1 and RGMII2 using MDIO1.

The Cortex M3 software controls the HW-RTOS GMAC, the 5-Port Switch, and the RGMII/RMII Convertors used by the 5-Port Switch. The Cortex M3 controls the external PHYs connected to the 5-Port Switch. The HW-RTOS GMAC is connected to the 5-Port Switch's upstream (in) port. See Figure 2 for details.

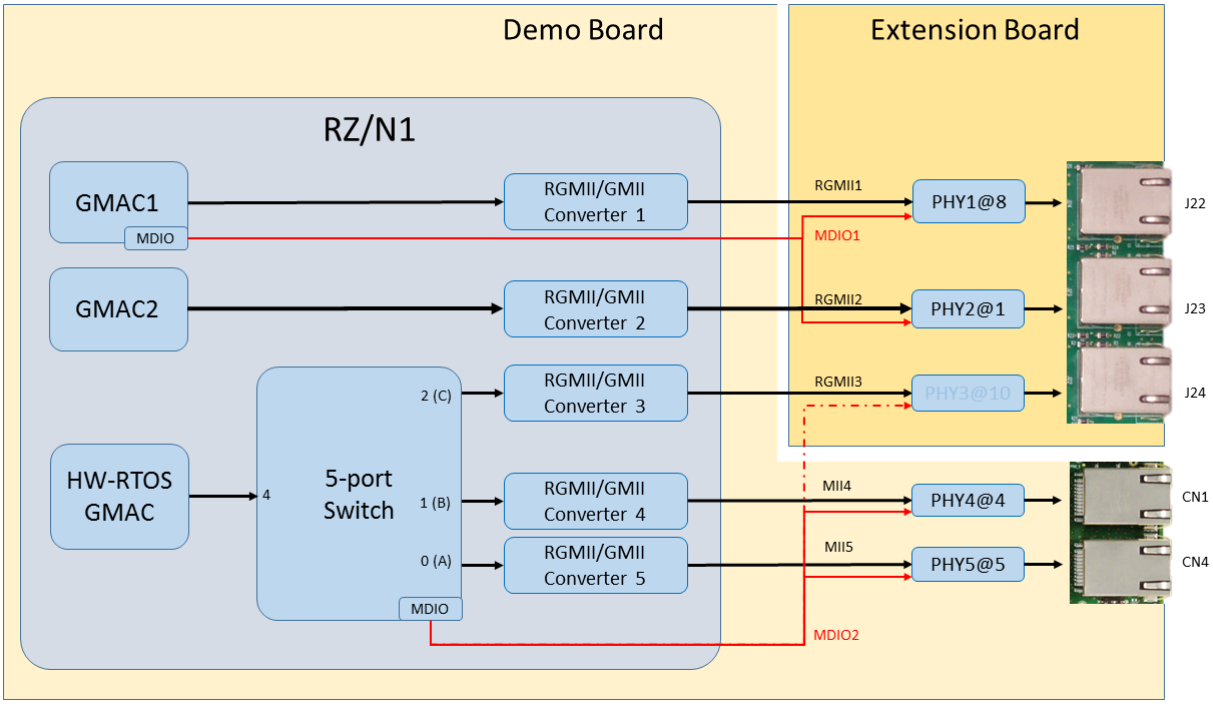

Figure 2 Master Mode Ethernet

This mode requires different jumper settings on the Extension Board:

- CN15 PHY2/PHY3 MDC Connect pins 1 and 2 (MDC1)
- CN16 PHY2/PHY3 MDIO Connect pins 1 and 2 (MDIO1)

Note: PHY3 is not accessible via MDIO in this hardware configuration.

GMAC1 is accessed via eth0, GMAC2 is accessed via eth1. An example of initialising Ethernet is:

```
ifconfig eth0 192.168.1.50 up
ifconfig eth1 192.168.1.51 up
ping -I eth0 192.168.1.30
ping -I eth1 192.168.1.30
```

When using the RZ/N1D-DB board without the Extension Board, there is no PHY connected to GMAC1 so eth0 is unusable.

4.1.3 No CM3 Mode

Device Tree Source: rzn1d400-db-no-cm3.dts

This configuration is the same as the Normal Mode, except Cortex M3 software must not use Ethernet. Linux controls GMAC1, GMAC2, the 5-Port Switch, the RGMII/RMII Convertors and all PHYs. This configuration is provided only as an example of how Linux could be used on its own. See Figure 3 for details.

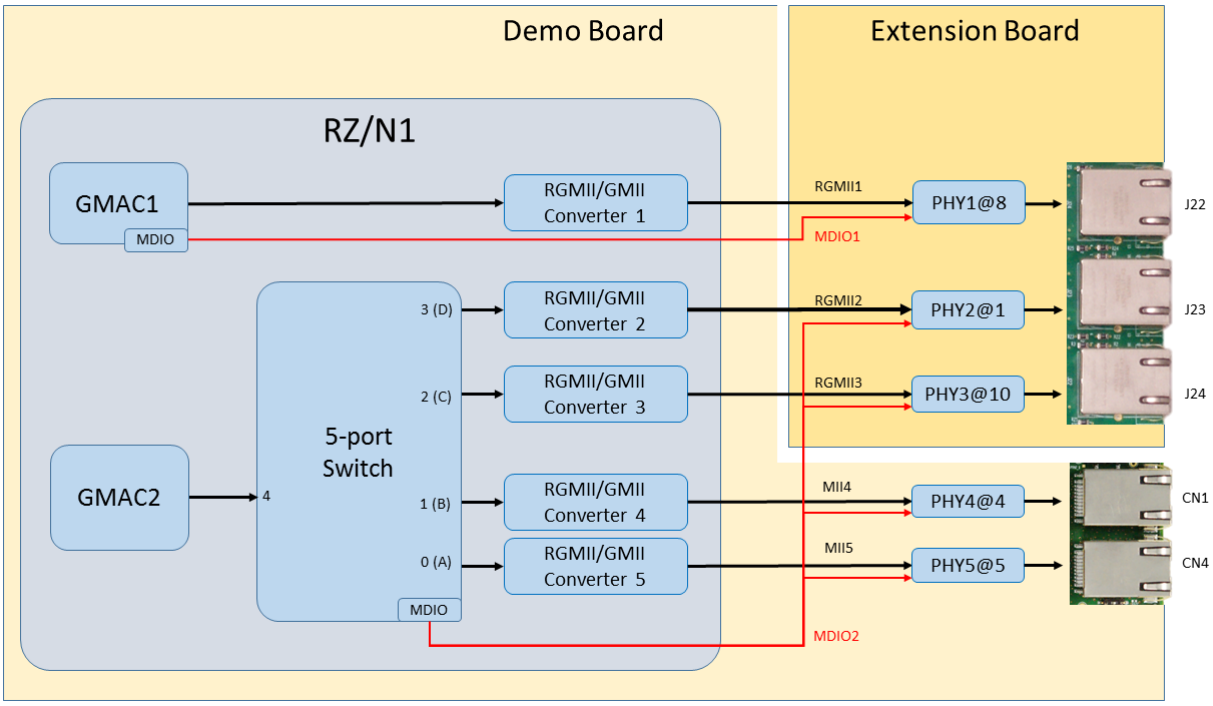

Figure 3 No CM3 Ethernet

If the RZ/N1D-DB board is used with the Extension board, GMAC1 is connected to RGMII1 to a PHY on the Extension Board, GMAC2 is used with the 5-Port Switch. Linux creates Ethernet network devices for the Synopsys GMACs and each of the Switch's downstream ports. The Synopsys GMAC devices are called eth0 and eth1, the Switch ports are sw0p0 through to sw0p3.

The network devices for the Switch ports must be enabled to use them, but cannot be used to transmit or receive data, all data is sent and received via GMAC2. These Switch devices allow you to talk to individual PHYs and set the MTU for each downstream port. You can use `ifconfig` to enable and disable the downstream ports by bringing the network device up or down, e.g.:

```
ifconfig sw0p0 up
```

The rzn1d400-db-no-cm3.dts specifies that only the downstream Switch port associated with CN1 (sw0p1) is enabled during boot.

An example of initialising Ethernet is:

```
ifconfig eth0 192.168.1.50 up
ifconfig eth1 192.168.1.51 up
ifconfig sw0p0 up

ping -I eth0 192.168.1.30
ping -I eth1 192.168.1.30
```

When using the RZ/N1D-DB board without the Extension Board, there is no PHY connected to GMAC1 so eth0 is unusable.

Both the Synopsys GMAC driver and the MoreThanIP 5-port Switch driver support changing the PHY settings using mii-tool and ethtool.

For example, to force the link setting of Switch port 2 (C) on the RZ/N1D-DB board using mii-tool:

```
mii-tool -F 100baseTx-FD sw0p2
```

For example, to force the link setting of Switch port 2 (C) on the RZ/N1D-DB board using ethtool:

```
ethtool -s sw0p2 speed 100 duplex full autoneg off
```

4.2 Starting Linux

The following are examples of typical kernel command line options (set as the bootargs U-Boot environment variable) used for NFS mounted, SD card, and QSPI rootfs respectively:

```
setenv rfs  "root=/dev/nfs rw nfsroot=${serverip}:/tftpboot/rzn1,v3"
setenv rfs  "root=/dev/mmcblk0p1 rootfstype=ext4 rw"
setenv rfs  "root=/dev/mtdblock7 rootfstype=squashfs init=/init"

setenv bootargs "console=ttyS0,115200 ${rfs} rootwait"
```

The uImage and dtb can be downloaded from a TFTP server, for example:

```
tftp  0x80008000 uImage
tftp  0x8ffe0000 rzn1d400-db.dtb
bootm 0x80008000 - 0x8ffe0000
```

Note: Linux will automatically turn off any clocks on the RZ/N1 that are not used by the Linux drivers. Therefore, if you are running software on the Cortex M3 core we recommend you disable this feature by adding "`clk_ignore_unused`" to the bootargs variable.

4.3 GPIOs

One common problem is finding the Linux GPIO number associated with the GPIO hardware. Each Synopsys GPIO Controller block has a driver that supports the multiple hardware ports. The Linux GPIO number associated with each port can be obtained from the sysfs interface.

A handy one-liner to list all of the GPIO ranges is:

```
for f in `ls -d /sys/class/gpio/gpiochip*`; \
do echo $f `cat $f/label $f/base $f/ngpio` ; done
```

This gives output similar to:

```
/sys/class/gpio/gpiochip366 5000d000.gpio 366 10
/sys/class/gpio/gpiochip376 5000d000.gpio 376 32
/sys/class/gpio/gpiochip408 5000c000.gpio 408 32
/sys/class/gpio/gpiochip440 5000c000.gpio 440 32
```

The entries may change if a different set of GPIO nodes is enabled in your Device Tree file. Similarly, if you use GPIO port expanders you may get additional entries. By looking up the address of the GPIO IPs, this shows that:

- gpio2b uses GPIO numbers 366 to 375
- gpio2a uses GPIO numbers 376 to 407
- gpio1b uses GPIO numbers 408 to 439
- gpio1a uses GPIO numbers 440 to 471

You then must manually map the Synopsys GPIO Controller to the pin number. The GPIO numbers will change if you have other sources of GPIOs on your board, like IO expanders – there is no way to have static numbers in this version of the Linux kernel.

4.3.1 GPIO Interrupts

ON the RZ/N1 devices, the interrupts from the GPIO Controllers are passed to a "GPIO interrupt multiplexer" block. This block then selects the interrupts that are passed to the GIC and NVIC interrupt controllers. This means you cannot have more than 8 GPIO interrupts. The GPIO interrupts must be added to the "gpioirq" node in your Device Tree file.

4.4 Supported Features

The following features are supported.

Hardware Block	Linux device	Interface/Device	Details
UART1	/dev/ttyS0	UART (CN10)	9600 to 115200, 1Mbps
UART3	/dev/ttyS1	Extension board: UART (J7)	9600 to 115200 (RS-232 Line Transceiver is limited to 250kbps).
UART4	/dev/ttyS2	Extension board: UART (J8)	9600 to 115200, 1Mbps, (RS-485 Line Transceiver is limited to 1Mbps).
QSPI	/dev/mtd0..7	Macronix MX25L25635F	Single, dual and quad modes. 62.5MHz SPI clock.
SDHC	/dev/mmcblk0*	Extension board: SD Card slot (J4)	50MHz SD clock. SD cards, SDIO Wi-Fi
I2C2	/dev/i2c1	Extension board: LM75 compatible Temp Sensor	At 400KHz I2C clock.
USB Host	/dev/sd*	Extension board: USB Host connecter (J20)	USB Mass Storage devices
USB Function (Device)	See network devices below	USB Function connecter (CN9)	RNDIS
GPIO	/sys/class/gpio	LEDs and switches	See Section 4.5.1
RTC	N/A	N/A	Accessed using hwclock command
SPI FRAM	/sys/bus/spi/devices/spi32766.0/eeprom	Extension board: FM25V10-G Serial (SPI) F-RAM	Limited to 20.8MHz SPI clock due to board design.
DDR	/sys/devices/system/edac/mc/mc0	DDR3	ECC error reporting and injection

The software provides the following network devices.

Hardware Block	Linux Network device	Interface/Device	Details
CAN1	can0	Extension board: CAN connector (J16)	100Kbps, 500Kbps and 1Mbps bitrates
USB Function	usb0	USB Function connecter (CN9)	RNDIS

In addition, the different Ethernet network devices are available depending on the mode of operation.

Normal Mode

Hardware Block	Linux Network device	Interface/Device	Details
GMAC1	eth0	Extension board: Marvell 88E1512 PHY (IC2) used with connecter J22	10, 100 Mbps and 1Gbps at full and half duplex.

Master Mode

Hardware Block	Linux Network device	Interface/Device	Details
GMAC1	eth0	Extension board: Marvell 88E1512 PHY (IC2) used with connecter J22	10, 100 Mbps and 1Gbps at full and half duplex.
GMAC2	eth1	Extension board: Marvell 88E1512 PHY (IC4) used with connector J23.	10, 100 Mbps and 1Gbps at full and half duplex.

No CM3 Mode

Hardware Block	Linux Network device	Interface/Device	Details
GMAC1	eth0	Extension board: Marvell 88E1512 PHY (IC2) used with connecter J22	10, 100 Mbps and 1Gbps at full and half duplex.
GMAC2	eth1	N/A	Fixed internal 1Gbps link to the upstream port of the 5-port Switch.
5-port Switch	sw0p0	5-port Switch port 0 (A) to Micrel KSZ8041TL (U12) PHY used with connector CN5.	100 Mbps full duplex only.
	sw0p1	5-port Switch port 1 (B) to Micrel KSZ8041TL (U8) PHY used with connector CN4.	100 Mbps full duplex only.
	sw0p2	5-port Switch port 2 (C) to Extension board Marvell 88E1512 PHY (IC4) used with connector J23.	10, 100 Mbps and 1Gbps at full and half duplex.
	sw0p3	5-port Switch port 3 (D) to Extension board Marvell 88E1512 PHY (IC3) used with connector J24.	10, 100 Mbps and 1Gbps at full and half duplex.

4.5 Accessing the hardware

4.5.1 LEDs

The LEDs can be accessed from the LED sysfs interface. Each LED in Device Tree has a label, and that label is a directory under /sys/class/leds. The supplied Device Tree uses the red and green LEDs on the expansion board (D20 and D21) for CPU activity.

To control these LEDs yourself, first replace the default trigger with "None" to disconnect the LED from the kernel use, for example:

```
echo none > /sys/class/leds/pl_gpio92/trigger
```

You can then set the LED on or off, for example:

```
echo 1 > /sys/class/leds/pl_gpio92/brightness
echo 0 > /sys/class/leds/pl_gpio92/brightness
```

If you want to re-attach the LED to a kernel trigger, you can show the available triggers by:

```
cat /sys/class/leds/pl_gpio92/trigger
```

Set the kernel trigger as heartbeat:

```
echo heartbeat > /sys/class/leds/pl_gpio92/trigger
```

4.5.2 Temperature Sensor

The LM75 device is connected via an I2C interface. This device is exposed to userspace via the Hardware Monitoring interface.

Using this interface, you can read the temperature in hundredths of degrees by running:

```
cat /sys/class/hwmon/hwmon0/temp1_input
```

4.5.3 EEPROM

The EEPROM device is connected via an I2C interface. You can read and write data to EEPROM using the `/sys/bus/i2c/devices/1-0050/eeprom` file. This file is a block device, so the maximum block size that can be used to access it is 4096 bytes. Here are some handy ways to access EEPROM. The following examples assume that this env variable is set:

```
EEPROM=/sys/bus/i2c/devices/1-0050/eeprom
```

Read the entire contents and display it:

```
cat ${EEPROM} | od -x
```

Clear the entire contents:

```
cat /dev/zero > ${EEPROM}
```

Read an exact amount of data:

```
dd if=${EEPROM} bs=64 count=1 | od -x
```

Write an exact amount of random data:

```
dd if=/dev/urandom of=${EEPROM} bs=64 count=1
```

4.5.4 FRAM

The FRAM device is connected via a SPI interface. You can read and write data to FRAM using the `/sys/bus/spi/devices/spi1.0/eeprom` file. Here are some handy ways to access FRAM. The following examples assume that this env variable is set:

```
FRAM=/sys/bus/spi/devices/spi1.0/eeprom
```

Read the entire contents and display it:

```
cat ${FRAM} | od -x
```

Clear the entire contents:

```
cat /dev/zero > ${FRAM}
```

Read an exact amount of data:

```
dd if=${FRAM} bs=64 count=1 | od -x
```

Write an exact amount of random data:

```
dd if=/dev/urandom of=${FRAM} bs=64 count=1
```

4.5.5 RNDIS

By default, the RZ/N1 kernel is configured such that it implements an RNDIS USB Device.

The information for assigning a static IP address to the RNDIS device can be found in /etc/network/interfaces, e.g.

```
iface usb0 inet static
                address 192.168.7.2
                netmask 255.255.255.0
                network 192.168.7.0
                gateway 192.168.7.1
```

To bring the interface up, run:

```
ifup usb0
```

You can now connect to the board from a Windows PC, e.g. if the RNDIS Adapter is assigned IP address 192.168.7.10, you can ping the board:

```
ping –S 192.168.7.10  192.168.7.2
```

5 ACRONYMS

AXI	Advanced eXtensible Interface (ARM bus)
CAN	Controller Area Network
CPU	Central Processing Unit
DDR	Double Data Rate (memory)
DFU	Device Firmware Upgrade
DMA	Direct Memory Access
DMAC	DMA Controller
DT	Device Tree
DTB	Device Tree Blob
DTS	Device Tree Source
eMMC	embedded Multi-Media Controller
ECC	Error Correction Code
FIFO	First In First Out (buffer)
FRAM	Ferroelectric RAM
Gb	Gigabit
GIC	ARM Generic Interrupt Control
GMII	Gigabit MII
IRQ	Interrupt ReQuest
I2C	Inter-Integrated Circuit
ISR	Interrupt Service Routine
MAC	Media Access Controller
MII	Media Independent Interface
MMC	Multi-Media Card
NFC	NAND Flash Controller
NFS	Network File System
OOB	Out Of Band
OTP	One Time Programmable
PLL	Phase Locked Loop
QSPI	Quad SPI
RGMII	Reduced GMII
RTC	Real Timer Clock
Rx	Receive
SD	Secure Digital
SDHC	Secure Digital Host Controller
SDIO	Secure Digital IO
SMP	Symmetric Multi-Processing
SPI	Serial Peripheral Interface
SPL	Secondary Program Loader
TFTP	Trivial File Transfer Protocol
Tx	Transmit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
XiP	Execute in Place

6 REFERENCES

Renesas RZ/N1 Group User's Manual, R18UZ0049EJ0000

Renesas RZ/N1 U-Boot User Manual, R01US0296EG

U-Boot, http://www.denx.de/wiki/U-Boot

Linux, https://www.kernel.org

Yocto, https://www.yoctoproject.org

BuildRoot, https://buildroot.org

7 CHANGE HISTORY

Version	Description	Date
1.0	First public release	13th Apr 2017
1.01	Minor corrections	11th July 2017
1.02	Yocto instructions updated. Replaced initramfs with SquashFS. DTB address moved to top of DDR. Minor corrections	18th Oct 2017
1.03	Moved to external git repo. I2C bus number changed due to use of alias in DT. Added RT-Linux instructions. Updated comment regarding RGMII/GMII Convertor handling.	26th Jan 2018
1.04	Corrected build instructions. Added details on DMAC bindings. Added detail on UART DMA. Updated Watchdog driver information. Fix SysCtrl register name. Removed MSEBI driver. Corrected menuconfig information for networking drivers. Removed redundant acronyms. Added Master Mode (2xGMACs) information. Changed Dynamic Frequency Control menu selection. Changed 5-Port Switch network device names. Yocto updated from Morty v2.2 to Rocko v2.4. Corrected GMII to RMII. Added note about GPIO interrupts.	27th Jun 2018
1.05	Changed build setup instructions to refer to exact version. When used with NFS root file system, specify NFSv3	11th Oct 2018
1.06	Fix filename for patch that allows Qt to be built without OpenGL. Improve description of different Ethernet configurations.	6th Nov 2018
1.07	Fix Yocto initramfs instructions.	12th Feb 2019
1.08	Fix Yocto build error by specifying the exact version of repos	20th Feb 2019
1.09	Removed NetDisp Yocto image. Add information about git merge conflicts	15th Jul 2019
1.10	Updated for Linux kernel v4.19	30th Aug 2019

RZ/N1 Linux

Renesas Electronics Corporation

R01US0297EG0110