RZ/N1D

System Setup Tutorial

R01ANxxxxEU
Rev.1.14
Jan 15, 2018

Introduction

The goal of this guide is to act as a hands-on tutorial to bring-up, design/debug software, and the maintainenance of an RZ/N based system. This guide discusses technical aspects related to tools, system maintenance, application development, debugging, and how to implement third party software. The main value of the document is explaining, again with the other documentation as a first reference, more in-depth engineering detail.

Together with the U-Boot and Linux Quick Start Guides of the YCONNECT-IT-RZN DVD image, the Setup Tutorial is a support document. The goal with this document is to not duplicate information. Where information syntax already exists, for example in DVD documentation, this is info is referenced and not duplicated here.

This document also comes with a build environment *build.sh* which provides a straightforward approach to creating and customizing a system. The following topics will be discussed, and more.

- U-Boot
- RZ/N1D linux kernel
    - Cross compiling
- Creating a root file system, to include
    - Basic linux command set
    - A7 core application(s)
- CM3 core applications
    - Building on Windows and adding to system
    - Debugging
- Using SD card as file system media
- Running a complete dual core system

The main documentation for the system is in the YCONNECT-IT-RZN DVD image. See References.

References

- RZ/N1-1 homepage
- YCONNECT-IT-RZN DVD
    1. Click on 'Downloads'
    2. You should see a DVD image link.
    3. Download all the files, then assemble the ISO image by executing the .exe file, then you should be able to unzip it.
- See also Documents and Software for how to download the latest documents and software.
- The “QSG” referenced in this document refers to the quickstart documents of the DVD, in particular .*\YCONNECT-IT-RZN\Documents\RZN.Software\U-Boot-and-Linux\RZN1D-DB-Quick-Start-Guide.pdf*.

Target Device

The RZ/N1D, mounted on the RZN1D-DB base board, and the add-on expansion board RZN1D-EB.

Contents

1. RZ/N1 Boards

This tutorial assumes you have a RZ/N1D-EB board, the large base board, and a CPU board RZ/N1D-DB. (The latter has the RZ/N1 chip with DDR memory and QSPI flash on board.)

1.1 Power

Mount the DB onto the EB board, and power the EB board with an extrnal adapter. You <u>must</u> use a 5 or 12/24 V DC adapter if you are using the EB board. Set the jumper CN14 accordingly.

CAUTION

- Using USB power will not be enough if you are using the EB board. You won't notice this if the USB power jumper is set for USB power. The board will reboot now and then and you won't know why.

- Some adapter plugs' center hole is a bit too large, causing power dropouts. If you don't have a proper adapter, you can try squeezing the adaptor plug a bit to make it connect with the center hole.
- IF you set CN14 the to use a 5 V adapter, do not use the 12/24 V adapter that comes with the kit.

2. Documents and Software

2.1 The DVD

The solution kit DVD can be found from the RZ/N page. Go to *Downloads* and look for the CONNECT IT ETHERNET RZ/N ISO-image. The link to the DVD is under References above. Download and assemble the ISO image as shown on that page, then you should be able to just unzip it to your local drive.

2.2 rzn1_linux-4.9_bsp

The package *rzn1_linux-4.9_bsp* package, with build-scripts for the RZ/N A7 linux and application code, and which downloads and install cross compiler, Buildroot, etc… can be downloaded via an REA SFTP access.

Host: tahoe.america.renesas.com
Protocol: Secure FTP (SFTP/SSH)
Port: 2200
Username: ren_customer
Password: 47#GreenSpider

A. Filezilla

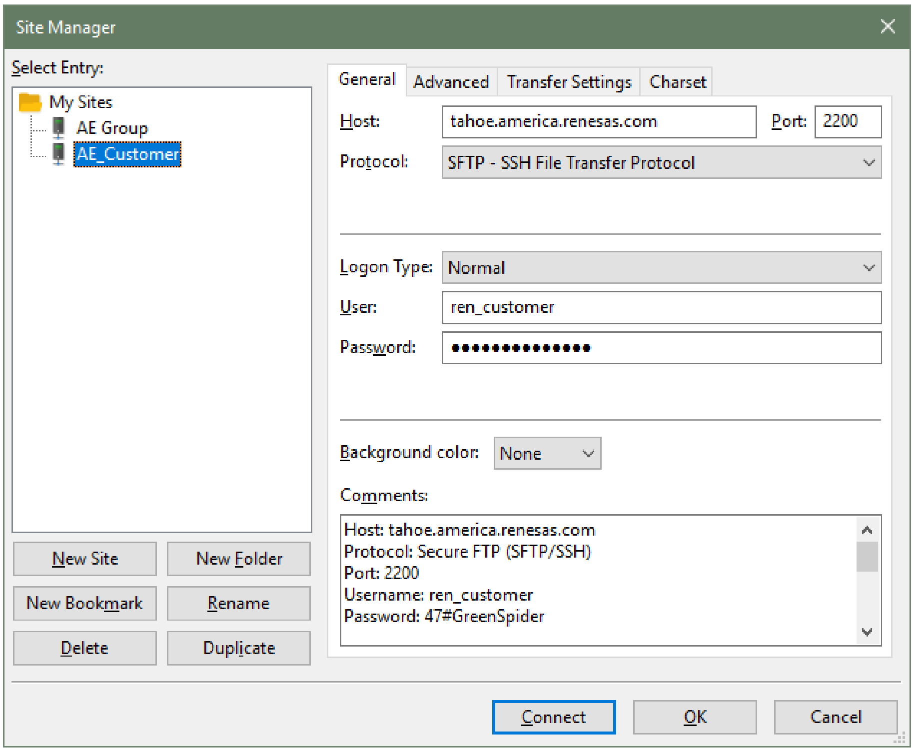


Figure 1. Customer FTP access to rzn1_linux-4.9_bsp.

B. HTTPS

https://tahoe.america.renesas.com
ENGG
ren_customer
47#GreenSpider

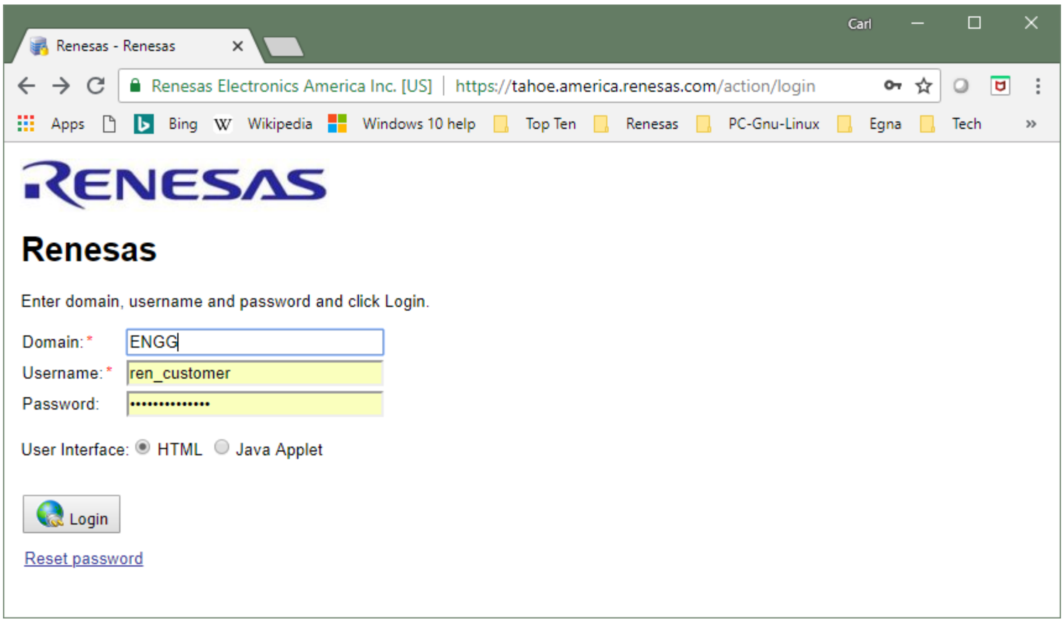


Figure 2. HTTPS interface to access to rzn1_linux-4.9_bsp. Same password as in Figure 1.

3. Windows

The CM3 core of the RZ/N1 has the R-IN Engine; the industrial networking features. Software for this core is compiled and debugged under Windows using the I-Jet download/debugger tool.

3.1 IAR IDE

Download the development environment for the Cortex CM3 code by and installing IAR-EWARM, version 8.11 as of this doc version Rev.1.12, by registering for a 30-day evaluation license from www.iar.com. The 30 days is with unlimited memory. The limited memory evaluation version is for an unlimited period.

4. Linux Host PC

Install Linux onto a host PC. Use Ubuntu, the 16.04 LTS release. Kubuntu is an alternative with slightly fancier user interface. https://kubuntu.org/getkubuntu

You will want to have both Linux and Windows PCs running without the need to reboot frequently. Therefore, choose how you want to work:

A) Typically, you can use an older laptop as the Linux host. You can easily make an old Windows laptop a dual-boot machine.

B) Install a virtual linux host onto your Windows machine, e.g. VirtualBox. This will take longer time to set up than A.

4.1 Needed Packages

Besides editor and other necessities for your linux host, install *gtkterm* and *dfu-util*. This enables host communication with the RZ/N1D-EB board. Also add packages *ncurses*, *lzod*, which will be needed later to build file system and binaries for the RZ/N.

```
$>sudo apt-get install gtk-term dfu-util ncurses lzod
```

Or, use your favorite package manager to install this and other needed packages. *Synaptic* package manager has an advanced user-friendly interface.

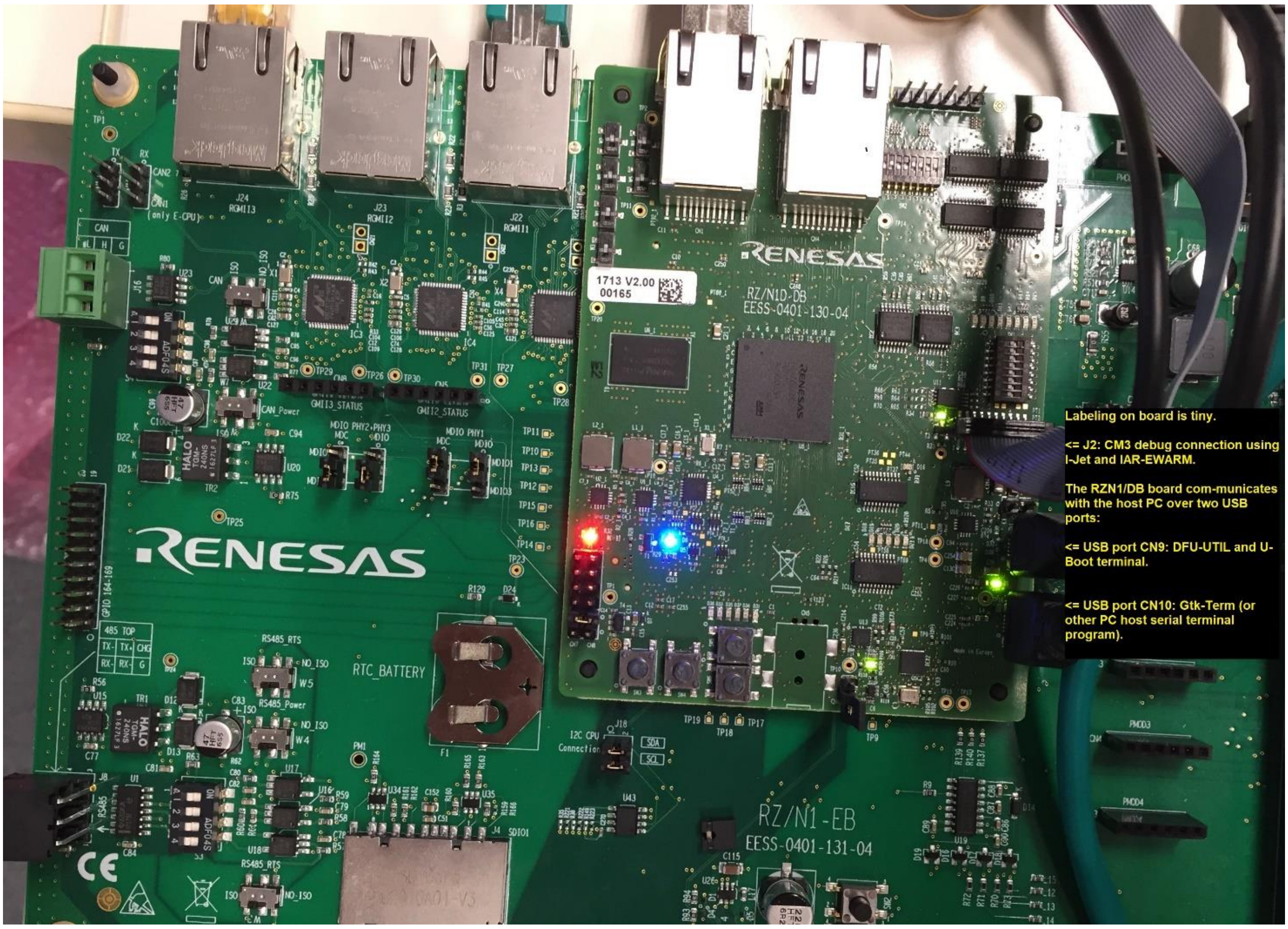


Figure 3. The upper USB connector on the CPU board is CN9 which will communicate with *dfu-util* tool of the Linux host. The lower USB connector, CN10, will communicate with your serial terminal, e.g. *gtkterm* of the Linux host. It should come up as the third enumerated USB port (of four) when it is plugged in to the PC, e.g. */dev/ttyUSB2* in Linux.

4.2 Connect to Board

Download the DVD content via SFTP as described in section 2 to your host PC.

Open the document *..\YCONNECT-IT-RZN\Documents\RZN.Software\U-Boot-and-Linux\RZN1D-DB-Quick-Start-Guide.pdf.*

The board jumpers J2, and switches W-1 to W-6 should be set as in the QSG guide. (Yellow markings.) Power the board with a power supply (5 V) and connect both USB micro connectors CN9 & 10 to your host PC. You should find it as the third (of four) newly enumerated USB devices, e.g. */dev/ttyUSB2*. Find the board prompt using for example *gtkterm*. Connect at 115200 bps. See Figure 4.

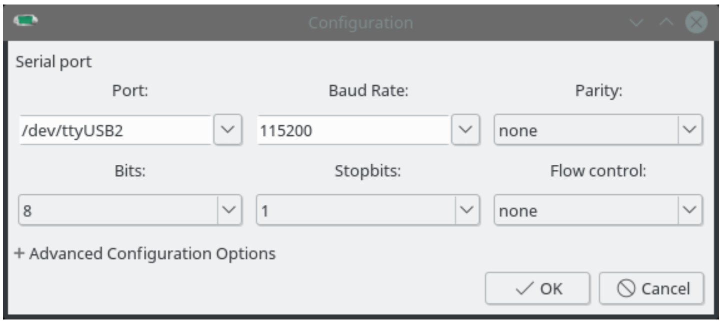


Figure 4. *gtkterm* settings to communicate with
- U-boot
- RZ/N1 linux serial terminal / command line
- Serial output from the CM3 program

5. U-boot

U-Boot is a secondary bootloader. The internal (pre-) bootloader does not need to be added as it comes with the RZ/N. The internal bootloader invokes U-Boot after you have installed a valid SPKG U-boot image to the RZ/N1 flash.

U-Boot is a very flexible loader with many commands for quickly adapting to a changing world. Code moved to a new address? The IP address changed? You can modify such things without rebuilding the kernel or device tree.

5.1 Installation

Install U-boot to the RZ/N1D-EB board. Use the precompiled binary of the DVD, e.g.

```
..\YCONNECT-IT-RZN\Software\U-Boot-and-Linux\u-boot\binaries\u-boot-rzn1d400-db.bin.spkg
```

To program U-Boot you need hold SW5 down before pressing SW3. Only do this for programming U-boot. (For programming any other flash area just type *dfu* in the U-Boot terminal and the rest is from the host terminal.)

There are multiple steps here, follow the Quick Start Guide. If you have problems with dfu-util, see Appendix 12.

5.2 Boot and Run U-Boot

With U-boot installed you should be able to boot with SW4 (or SW3) and see the U-Boot prompt, as in Figure 5.

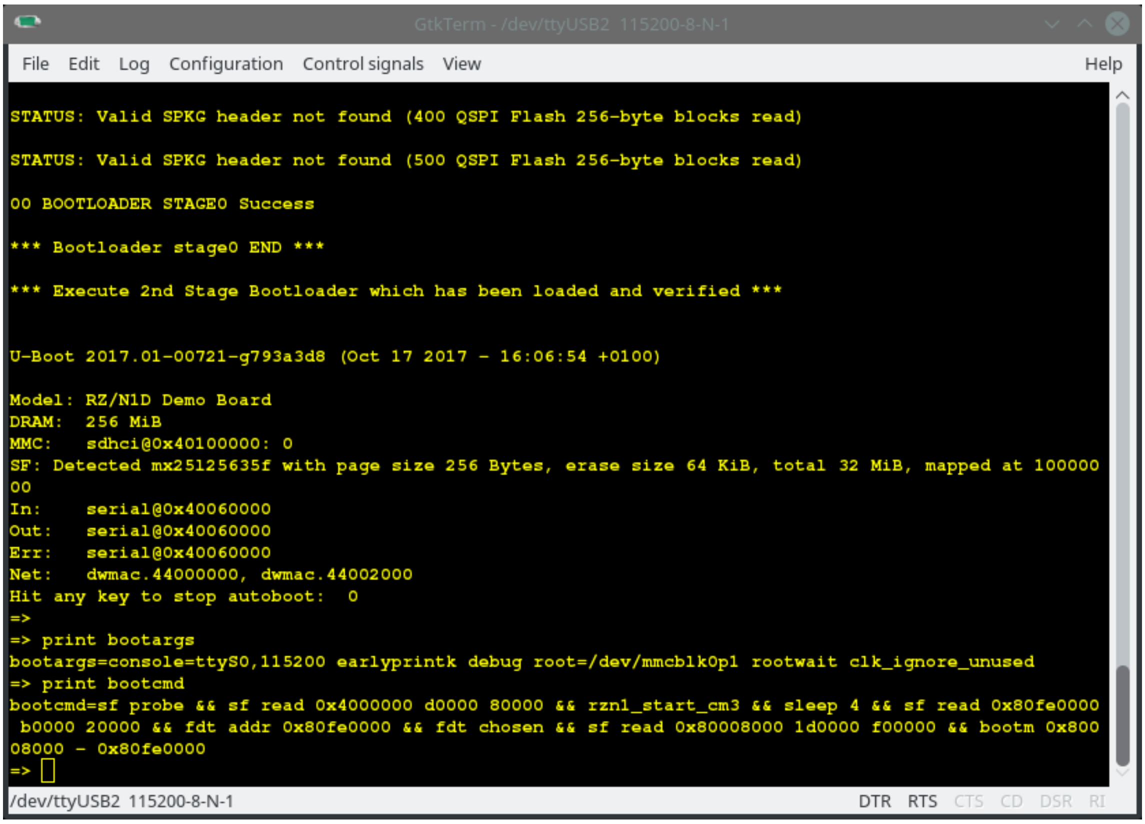
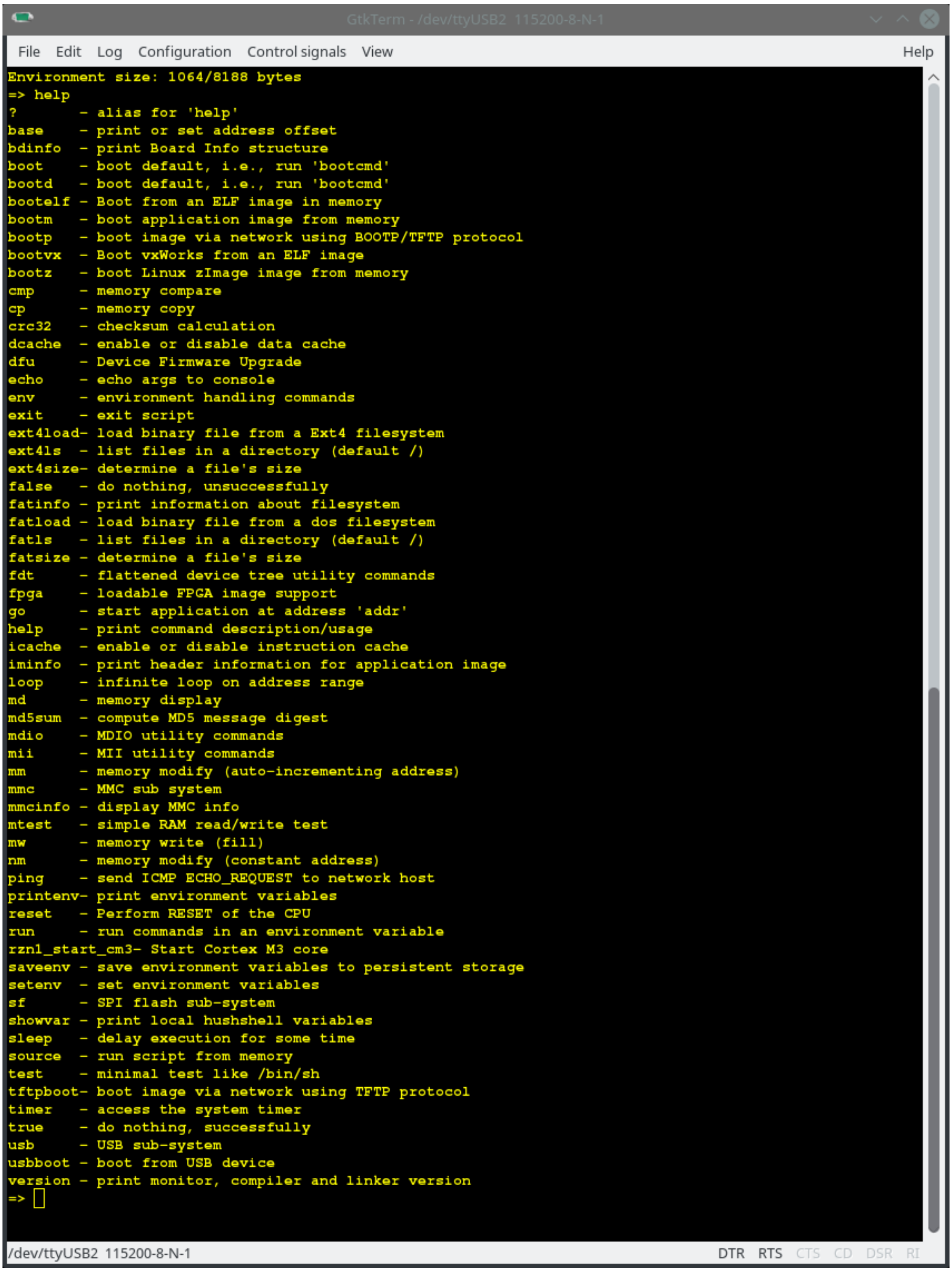


Figure 5. U-boot after booting and pressing any key to prevent autorun of the 'boot' command, then issuing a couple of u-boot environment variable print commands.

If you type *help* at the U-Boot prompt you will see the complete U-boot command set.




```
Environment size: 1064/8188 bytes
=> help
?       - alias for 'help'
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
bootz   - boot Linux zImage image from memory
cmp     - memory compare
cp      - memory copy
crc32   - checksum calculation
dcache  - enable or disable data cache
dfu     - Device Firmware Upgrade
echo    - echo args to console
env     - environment handling commands
exit    - exit script
ext4load- load binary file from a Ext4 filesystem
ext4ls  - list files in a directory (default /)
ext4size- determine a file's size
false   - do nothing, unsuccessfully
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
fatsize - determine a file's size
fdt     - flattened device tree utility commands
fpga    - loadable FPGA image support
go      - start application at address 'addr'
help    - print command description/usage
icache  - enable or disable instruction cache
iminfo  - print header information for application image
loop    - infinite loop on address range
md      - memory display
md5sum  - compute MD5 message digest
mdio    - MDIO utility commands
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - display MMC info
mtest   - simple RAM read/write test
mw      - memory write (fill)
nm      - memory modify (constant address)
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
rzn1_start_cm3- Start Cortex M3 core
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sf      - SPI flash sub-system
showvar - print local hushshell variables
sleep   - delay execution for some time
source  - run script from memory
test    - minimal test like /bin/sh
tftpboot- boot image via network using TFTP protocol
timer   - access the system timer
true    - do nothing, successfully
usb     - USB sub-system
usbboot - boot from USB device
version - print monitor, compiler and linker version
=>
```

Figure 6, **U-boot after issuing the help command. Type help <command> to see syntax for each command.**

The following commands are the most important, and those with which you should be somewhat familiar with what they are for.

boot - Boot default, i.e., run the 'bootcmd' environment variable as a boot script. Normally to start linux and/or the cm3 core application.

bootm - Boot application image from memory [need address as argument].

dfu - Device Firmware Upgrade.

env - Environment handling commands.

fdt - Flattened device tree utility commands.

help <command> - Print description/usage for ***command***.

loop - Infinite loop on address range.

mmc - MMC sub system commands.

mw - Memory write (fill).

ping - Send ICMP ECHO_REQUEST to network host.

run [x] - Run commands in an environment variable x.

rzn1_start_cm3 - Start Cortex M3 core

setenv - Set environment variables.

saveenv - Save environment variables to static memory storage [QSPI in our case].

sf - SPI flash sub-system, read and write [to QSPI].

sleep [x] - Delay execution for some time.

Type *help <command>* to see the syntax

5.3 SD/MMC

You can access an SD card from U-Boot if you have a formatted SD with an *ext4* linux file system.

```
=>mmcinfo
```

To list files on an SD.

```
=>ext4ls mmc 0:1
```

With a FAT file system this would be

```
=>fatls mmc 0
```

See further section *File System*.

5.4 Environment Variables to Run CM3 and Linux

U-Boot's capability to use environment variables makes it very powerful tool. Often, something changes in the surroundings that are temporary, that will vary with the application. For example, IP-address and root file system media.

5.4.1 bootcmd

bootcmd is a U-boot script with commands separated by '&&'. The script will run when you run the *boot* command.

Most common usage here would tyupically be:

setenv bootcmd "sf probe && sf read 0x4000000 d0000 80000 && rzn1_start_cm3 && sleep 4 && sf read 0x8ffe0000 b0000 20000 && sf read 0x80008000 1d0000 f00000 && bootm 0x80008000 - 0x8ffe0000"

The following table breaks this down and explains each part.

U-Boot Command	Meaning
sf probe	Start the serial flash tool **sf**.
sf read 0x4000000 d0000 80000	Read 0x8000 (all 32 k) bytes of CM3 area from QSPI @ 0xd0000 and load to SRAM at 0x40000000.

rzn1_start_cm3	Run the CM3 program from reset.
sleep 4	Wait for 4 seconds before running next command.
sf read 0x8ffe0000 b0000 20000	Read 0x20000 (all 128 k) bytes of DTB area from QSPI @ b0000 and load to SRAM at 0x8ffe0000.
fdt addr 0x8ffe0000	Tells U-boot where the linux DTB is in flash.
fdt chosen	Activates *fdt_chosen* structure of DTB instead of *bootargs*. Must be preceded with address of DTB; "*fdt addr 0x8ffe0000 &&*".
sf read 0x80008000 1d0000 f00000	Read 0xF00000 (all 16 M) bytes of kernel area from QSPI @ 0x1D0000 and load to SRAM at 0x80008000.
bootm 0x80008000 - 0x8ffe0000	Boot kernel image stored now in DDR at 0x80008000 and device tree at 0x8FFE0000.

5.4.2 bootargs

In many cases you will want to pass additional information to the RZ/N1 linux kernel; for instance, information about which root device to use, or the network configuration. In U-Boot this is supported using the *bootargs* environment variable. *bootargs* overrides the *fdt_chosen* structure of the device tree. Its content is passed to the Linux kernel at startup as command line arguments.

This allows the use of the same Linux kernel image in a range of configurations. For well-written information on this see http://www.denx.de/wiki/view/DULG/LinuxKernelArgs

5.4.3 Set bootargs and bootcmd

Set the U-boot environment variables *bootparams* and *bootargs* according to the QSG.

5.5 Boot to Linux

From QSPI

To load and run the linux OS, DTS, etc, type

=>boot

and execution will start from addresses given by *bootm* and with *bootargs* parameters which will be passed to the kernel.

SD

You can boot Linux from an SD card instead of from flash (QSPI). See 9.4.*Boot Linux from SD.*

6. RZ/N1 Linux

We will later see how to build a custom linux kernel, but for normal cases the precompiled binary is perfectly fine. It was created for the RZ/N1. This kernel and the device tree binary (DTB) are in the DVD.

```
..\YCONNECT-IT-RZN\Software\U-Boot-and-Linux\kernel\binaries\
```

6.1 Device Tree

A linux device tree is comparable to the BIOS of a PC. The source is a DTS-file. When compiled it is called a DTB, or "blob".

Load the RZ/N linux DTB to QSPI. Use the precompiled binary *uImage-rzn1d400-db.dtb* from the above *binaries* folder. Run *dfu* at the U-boot terminal together with *dfu-util* on a Linux host command line to download the DTB.

See QSG for syntax--if you have problems using DFU to program the RZ/N, see Appendix 12.

6.2 Kernel

uImage was built with the default RZ/N1 linux configuration, determined by the settings of file *rzn1_defconfig*. This file is tucked away in the linux kernel patch files, and only used should you build the kernel from source.

6.2.1 Write to system

If your bootargs include the string *root=/dev/mtdblock7* then your filesystem is mounted from SPI flash. If your bootargs have *root=/dev/mmcblk0p1* your filesystem will be mounted from an SD. See 9.2 for more on this.

6.2.2 Flash

Download to QSPI the precompiled RZ/N linux kernel *uImage* from the DVD. You can find it on the DVD at `./Linux\kernel\binaries\core-image-minimal-rzn1.squashfs`. The QSG section *Write Linux to QSPI* has the syntax.

6.2.3 SD

You can as an alternative load the kernel and DTB to an SD that contains your root file system, and load it from there instead. See 9.4.*Boot Linux from SD.*

6.3 File System

Later we will discuss building your own file system and loading it to an SD (MMC) card. For testing the system, a compressed, read-only, root file system image *squashfs* is available in the *binaries* directory. A Squash FS file system is intended for block-device memory common in embedded systems, like our QSPI, where low overhead is needed.

Download as described in the QSG to QSPI.

6.4 Boot to Linux

Boot to linux from the board by pressing SW4. This will first reboots to U-boot which in turn calls its *boot* command, after *bootdelay* seconds. You can issue *boot* yourself if you stop the autoboot mechanism by typing any character into the U-boot terminal before the *bootdelay* timeout. You may for example want to inspect or change your U-boot environment variables before booting the main system.

7. Cortex M3 Applications

All U-boot commands have so far been for the A7 core except for one. In *bootcmd* the following was included:

```
sf read 0x4000000 d0000 80000 rzn1_start_cm3
```

This was included to start up a Cortex M3 image in parallel to the A7 processing. These U-boot commands load 0x80000 bytes from QSPI at address 0xD0000, to DDR RAM at address 0x4000000. The CM3 core we must now therefore load to QSPI at 0xD0000.

7.1 Build CM3 image

After installing IAR-EWARM, open and build a CM3 application to get a binary CM3 executable.

Here we will use the Core2Core project “CTC” locate in

```
..\RZ-N\YCONNECT-IT-RZN\Software\GOAL\goal\projects\00410_goal\ctc_cc.
```

To open the workspace, double-click on the file

```
..\YCONNECT-IT-RZN\
Software\GOAL\goal\projects\00410_goal\ctc_cc\iar\7_70\rzn_demo_board_eb\rzn_demo_board_eb.eww
```

Compile (F7). The build output is in

```
..\RAM Debug - Eval Board\Exe..
```

Save the BIN-file to a USB stick. Copy it over to your Linux host PC.

7.2 Download

Use *dfu* of U-boot together with *dfu-util* of your Linux host to download a CM3 application to the board QSPI. The QSG has the syntax. If you have problems using DFU to program the RZ/N, see Appendix 12.

To run the dowloaded executable from QSPI at boot, add <*&& rzn1_start_cm3*> to u-boot's bootcmd, right after command to read the cm3 image to RAM, e.g:

```
=>setenv bootcmd "sf probe && sf read 0x4000000 d0000 80000 && rzn1_start_cm3 && sleep 4 && sf
read 0x8ffe0000 b0000 20000 && fdt addr 0x8ffe0000 && fdt chosen && sf read 0x80008000 1d0000
f00000 && bootm 0x80008000 - 0x8ffe0000"
```

or without fdt chosen

```
=>setenv bootcmd "sf probe && sf read 0x4000000 d0000 80000 && rzn1_start_cm3 && sleep 4 && sf
read 0x8ffe0000 b0000 20000 && sf read 0x80008000 1d0000 f00000 && bootm 0x80008000 - 0x8ffe0000"
```

The CM3 binary should now execute right after <*boot*> is issued from U-boot. And you should see traces from the CM3 output, each line starting with *CC_xxx*...

Reboot the system and check that the CM3 image executes right after U-boot's *boot* is issued. You will know if the CM3 image executes if you see output in the RZ/N1 console prefixed with [*CC_*]. See Figure 7.

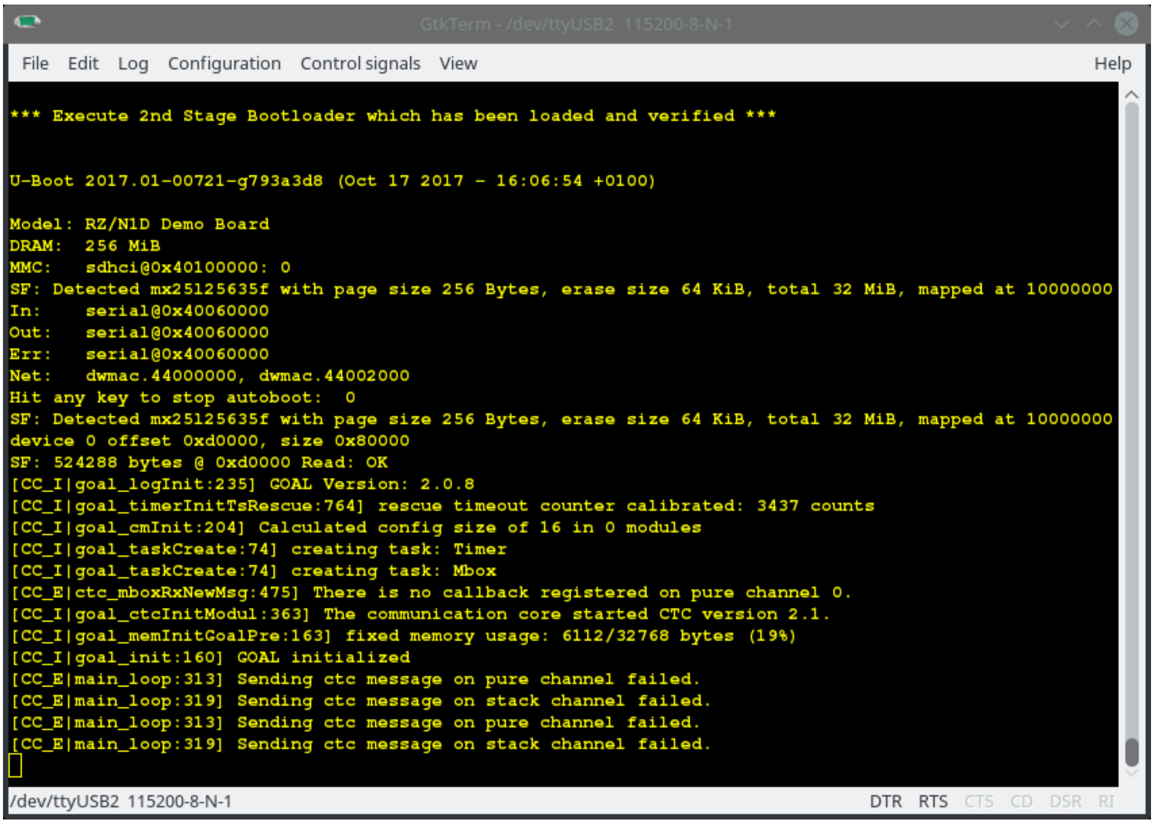


Figure 7. A CM3 Application is started by the U-boot commands *sf read 0x4000000 d0000 80000* and *rzn1_start_cm3*. In this case, the output "*Sending ctc message on pure channel failed*" is expected as the corresponding A7 core application has not yet started. That is started after under linux is up and running.

7.3 Debug

Use the IAR I-Jet to debug a CM3 project from Windows.

7.3.1 Debug CM3 standalone

On the -DB board, set switch W1 OFF (away from white bar), SW2-6 ON, SW2-7 OFF, SW2-8 ON. Observe that the IAR I-Jet debugger uses Arm Coresight Mode, determined by switch W1.

Release CM3 from reset by issuing from the U-Boot console

```
=>rzn1_start_cm3
```

Now you should be able to debug from IAR-EWARM.

Compile [F7].

Download and Debug [Ctrl+D].

7.3.2 Debug CM3 with application running on A7

Set -DB switches as in 7.3.

Set the following as *bootcmd*.

```
=>setenv bootcmd "mw 0x04000004 1 && rzn1_start_cm3 && sleep 5 && sf probe && sf read 0x8ffe0000
b0000 20000 && sf read 0x80008000 1d0000 600000 && bootm 0x80008000 - 0x8ffe0000"
```

Here is explanation for the new *bootcmd*.

To prevent the debug image to not be overwritten by the CM3 app stored on the board (right after the pause in *bootcmd*), we remove
~~***sf read 0x4000000 d0000 80000***~~

To setup/enable the CM3 core for the debugger, we add
mw 0x04000004 1

Set CM3 core to run
rzn1_start_cm3

Add a time window during which you can download an application from IAR-EW to subsequently debug it.
sleep 10

Load the CM3 binary via the IAR Debugger (Ctrl+D from IAR-EWARM) during the sleep period, that is, right after you have issued *boot* from U-Boot.

```
=>boot
```

With the code downloaded to the RZ/N board and waiting at reset, wait until linux is done booting (login as root) before running the code with ‘Go’ (F5).

8. Creating an A7 Linux Based System

U-boot and the Linux kernel have been customized for the RZ/N. Renesas provides patches for both. The Linux kernel and u-boot code are on the DVD, and are hosted on GitHub.

U-boot is version 2017.01 and supports all interfaces that are needed for booting - flash, Ethernet, SDHI, USB.

The Linux kernel is version 4.9. It supports all hardware IP blocks in the RZ/N that are outside of the RIN32 engine. The main build tool for Linux is the GCC toolchain. It can be hooked to an IDE of choice - e2studio, Eclipse, etc.

<u>Yocto</u>

The Yocto build environment is covered in separate Japan-issued documentation. See “RZ/N1 Linux System-on-Chip”. (Linux-User-Manual.pdf on the DVD.)

<u>Buildroot</u>

This document is based on Buildroot.

The build environment for U-Boot and Linux in this document is based on the package *rzn1_linux-4.9_bsp*. This package has a build a script *build.sh* which downloads above patches, builds U-Boot, Buildroot, Linux, and downloads, installs, and sets up a cross-compiler for the kernel and for your own linux applications, and more. See section 8.2.

8.1 Get and Unpack Build environment

REA will need to share *renesas-rz/rzn1_linux-4.9_bsp2* with you.

The repository for linux should be publicly available at https://github.com/renesas-rz/rzn1_linux.git

Dowload branch **rzn1-stable**.

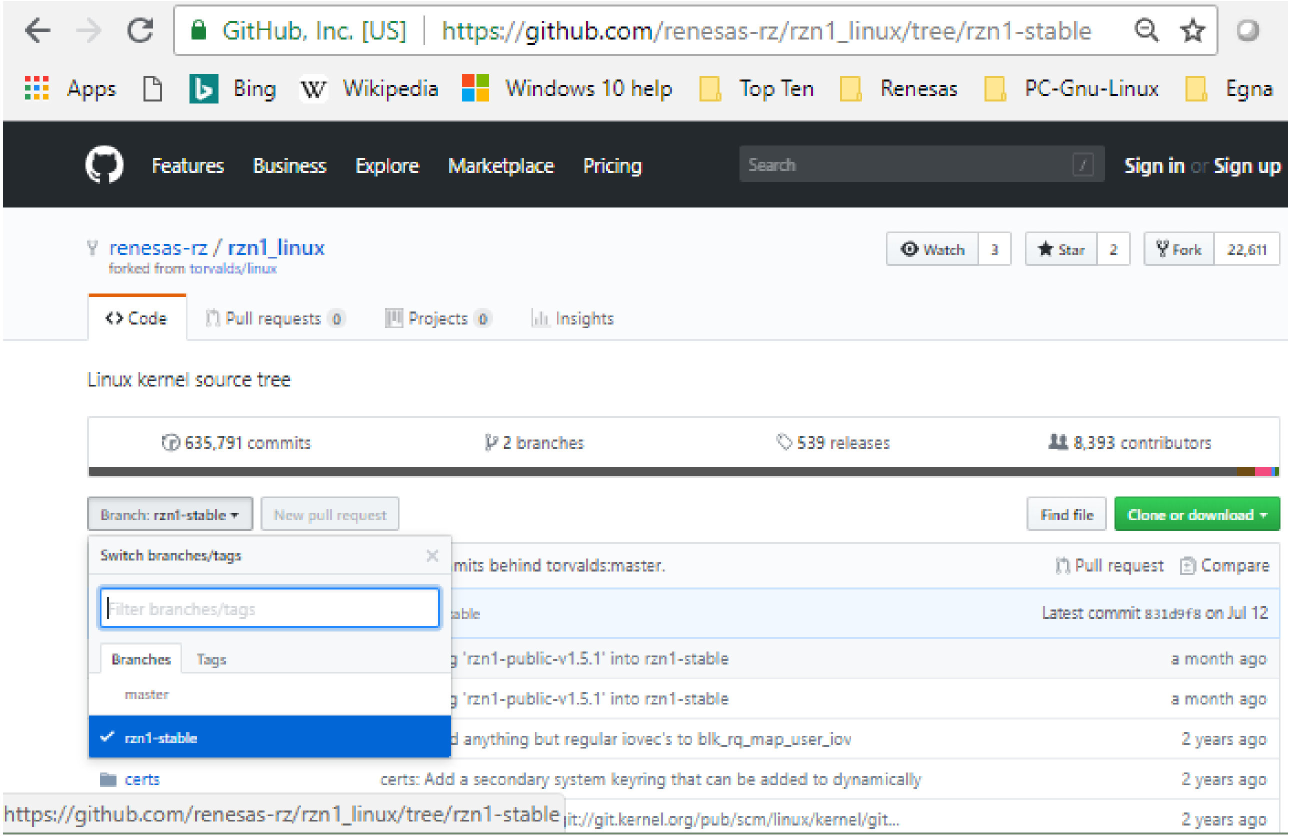


Figure 8. Getting the latest linux source for RZ/N.

rzn1_linux-4.9_bsp2 can download this, and do many other things for you. Unzip *rzn1_linux-4.9_bsp2.zip* to your Linux host PC.

You should see the following directories and files.

```
build.sh*
hello_world/
patches-buildroot/
patches-dfu-util/
patches-kernel/    ← not needed with rzn1_linux-4.9_bsp ver. 2.
patches-uboot/
setup_env.sh
```

Install and use *git* to track changes to your files. See a git quick start to initialize, add, and track changes you make.

8.2 build.sh - Universal RZ/N1 Builder

The build script *build.sh* can help you with many different tasks; such as downloading, building, and configuring the following.

- U-Boot.
- Buildroot. To create a root file system, incorporating Busybox to provide a small powerful linux command set.

- The linux kernel (based on the configuration *rzn1_defconfig*)
- Cross-compiler (*arm-linux-gnueabihf* by default) Sets you up to cross-compile code for the RZ/N A7 core with minimum effort.
- Host DFU utility

build.sh does the following automatically.

- Sets up build environment variables
- Downloads source code
- Applies patches to source code

Needed host packages

You may need to install more packages listed as missing when building the files system and cross-compiling. See 4.1.

To see a menu of how to use *build.sh*, type

```
$>cd rzn1_linux-4.9_bsp2
$>./build.sh
```

See Figure 9.

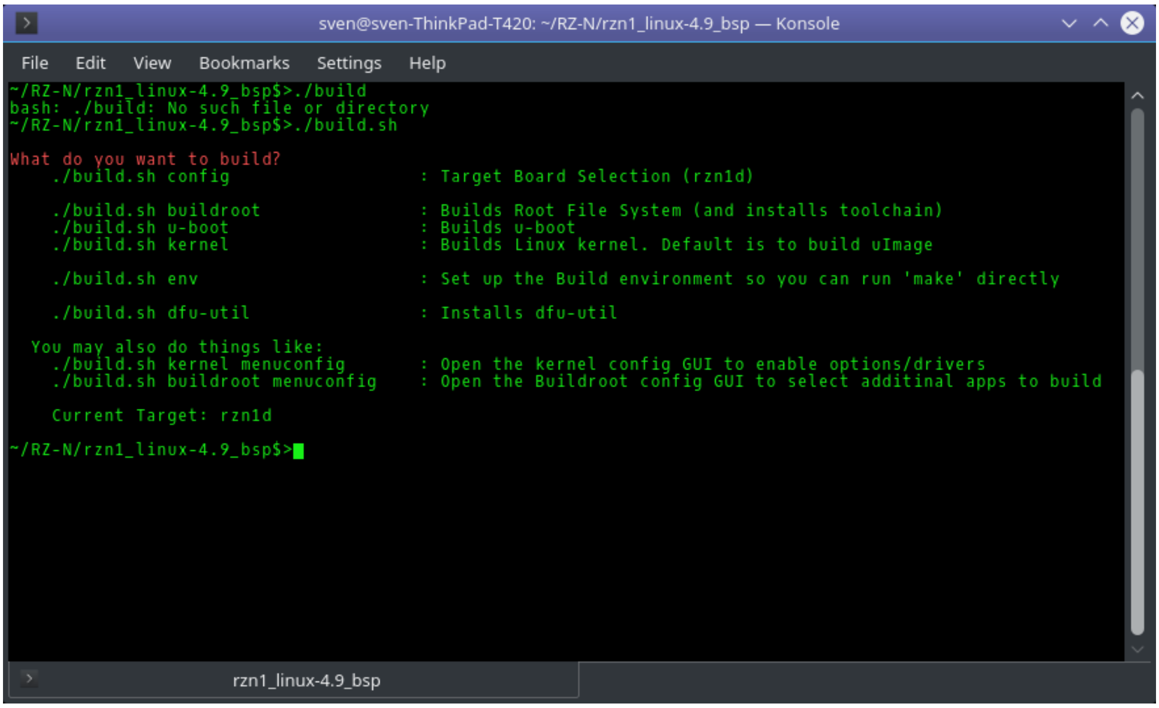


Figure 9. Running the build script with no arguments presents you with information on how to use.

Select Target

The script allows customization for different RZ/N1x variants. To select the variant, run

```
$>./build.sh config
```

8.3 Quick Summary

Here is a very brief summary of the rest of section 0. Thhis is what you would do in a normal workflow.

Root file system

Build root file system, including Busybox and user files in rootfs_overlay.

```
$cd ~/rzn1_linux-4.9_bsp2
$git init
$./build.sh config
$./build.sh buildroot menuconfig
$mkdir output/buildroot-2017.02/output/rootfs_overlay
$./build.sh buildroot
```

RZ/N kernel

```
$./build.sh kernel rzn1_defconfig
```

Exit and save. After this your configuration will be in the local .config file, and you will subsequently only need do the standard

```
$./build.sh kernel menuconfig
```

Change your kernel the way you want it.

```
$./build.sh kernel uImage
```

To update kernel with REA updates, should not have to be done often.

```
$./build.sh update k
```

This branch includes the GOAL Core-to-Core (C2C) driver from port GmbH. OBSERVE: The C2C driver version must match the GOAL Cortex M3 code version!

Add any files you change, e.g.

```
$git add <files>
$git commit
```

…

8.4 RZN1 Kernel

Leaving the linux kernel as-is should be fine to start with. It is already customized for the RZ/N1 variant you chose in 8.2. If you do want to change the kernel, issue the following.

```
$>./build.sh kernel rzn1_defconfig
```

The default kernel configuration is determined by file *rzn1_defconfig*. If you run above and do change the kernel configuration and save it, your configuration will instead be in the local *.config* file. After this you will always do

```
$>./build.sh kernel menuconfig
```

TIP: in the menuconfig tool, type '/' to search for configurable items of the kernel sourced code.

To build the uncompressed linux kernel, run

```
$>./build.sh kernel uImage
```

The kernel *uImage* binary is now in

```
../output/linux-4.9/arch/arm/boot/
```

Load it to the board just as you did for the prebuilt binary.

Latest kernel code

To update kernel with REA updates, should not have to be done often.

```
$./build.sh update k
```

This branch includes the GOAL Core-to-Core (C2C) driver from port GmbH. OBSERVE: The C2C driver version must match the GOAL Cortex M3 code version!

8.5 U-Boot

Building U-Boot

You should stick to the Renesas released binary to start with as it is tailored for the board and should be ok. If needed you can build U-boot (for example to change the serial port clock settings).

```
>$./build.sh u-boot
```

The output is in *./output/u-boot-xx.x/u-boot.bin*

You'll need to convert it to an SPKG image. See the Linux & U-boot guide, but here is example with an image named *u-boot-rzn1db400-rea-jan30.spkg*.

```
>$cd ~/RZ-N/rzn1_linux-4.9_bsp/output/u-boot/
```

```
>$~/RZ-N/YCONNECT-IT-RZN_V1.2/Software/U-Boot-and-Linux/u-boot/binaries/spkg_utility -i ./u-boot.bin -o ./u-boot-rzn1db400-rea-jan30.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
```

Download the new U-boot image to SRAM. (Hold SW5, press SW3.), for example

On the RZ/N serial terminal:

```
=> dfu
```

On the host PC:

```
$>sudo dfu-util -D ~/RZ-N/rzn1_linux-4.9_bsp/output/u-boot-xx.x/u-boot-rzn1db400-rea-jan30.spkg
```

If you have problems using DFU to program the RZ/N, see Appendix 12.

Flash it to serial QSPI (after issuing => *dfu* on the RZS/N again):

```
$>sudo dfu-util -a 'sf_uboot' -D ~/RZ-N/rzn1_linux-4.9_bsp/output/u-boot-xx.x/u-boot-rzn1db400-rea-jan30.spkg
```

Observe here that *'sf_spl'* is the first region of QSPI, and '*sf_uboot*' is the second. Use 'sf_spl' if your u-boot is not the one booted.

8.6 Device Tree

You can edit the device tree, but you should use the Renesas released binary to start with.

The most common reason is to change the *fdt_chosen* structure, which contains the default *bootargs* values. To edit it.

```
$>gedit output/linux-4.9/arch/arm/boot/dts/rzn1d400-db.dts
```

Build the DTS, to create your own DTB.

```
$>./build.sh kernel rzn1d400-db.dtb
```

*Replace with name of your editor.

8.7 Preconfigured Root File System of DVD

There is a default root file system on the DVD, that can be written to serial flash. The file system image is of type *squashfs*. If you want to use this preconfigured file system, see section 6.2.2 Flash. Also see the "Using bootargs' section below.

8.8 Custom Root File System

With the build script, you can build a Buildroot root file system. This will by default (unless you change as shown below) include Busybox; a small executable in linux that contains many common linux commands but with a smaller footprint. If you want to change something in Buildroot, issue the following, though leaving it as is for now should be OK, so you can skip this unless you want to add something as in the example "adding upload ability to the system" below.

```
$>./build.sh buildroot menuconfig
```

Doing this for the first time is equivalent to using *rzn1_defconfig*, which sets the default.

```
$>./build.sh buildroot rzn1_defconfig
```

Example; add upload ability to system

Let us add as an example XYZ modem capability as a binary program to the Buildroot tree. Using this we will be able to handily transfer new files to the target from a host PC. Mark the package *lrzsz* for inclusion to the tree:

```
Target packages  --->
  Networking applications  --->
    [*] lrzsz
```

How to use this is explained in 10.3 *Upload Files to Deployed System*.

Add your own files to the target file tree

Before building the root file tree, we need to add a folder that the script will look for, namely folder `rootfs_overlay`. This folder is where you will put anything you want included in the target tree. For example, if you want to add a file `/usr/bin/`**`myprog`** to the system, you would add it as e.g.

```
../buildroot-xx.x/output/rootfs_overlay/usr/bin/myprog
```

Therefore, you need to at least add the folder `rootfs_overlay/` next to `images/` or you will get an error from build.sh.

Build the file system

```
$>./build.sh buildroot
```

Specifically, we are interested in the *rootfs.tar* output. This is our compressed file tree.

```
~/RZ-N/rzn1_linux-4.9_bsp/output/buildroot-xx.x/output/images/rootfs.tar
```

We later need to load the file system to our target, the RZN1D-EB in our case. See next section

File System how to do this.

9. File System

Here we go over how to load a root file system, for example the *root.fs* we created in 8.7. We will focus on using an SD card as media. We created a file system using Buildrootm and using an SD it is easy to later add user files and programs to the file tree.

9.1 On SD

If the SD card is already formatted, erase everything.

```
$>sudo rm -fr /media/sven/rzn1_sdcard/*
```

SD cards come as FAT32, so we need to format it to native linus ext4 format. To do this, you can install *gparted* to your linux distro. Run it as root.

```
$>sudo gparted
```

Create one partition as an *ext4* partition.

Copy *root.fs.tar* to the SD card.

```
$>sudo cp ~/RZ-N/rzn1_linux-4.9_bsp/output/buildroot-xx.x/output/images/rootfs.tar
/media/sven/rzn1_sdcard
```

Unzip the root filesystem to the media

```
$>cd /media/sven/rzn1_sdcard/
$>sudo tar -xf rootfs.tar
```

Before removing from PC, make <u>sure</u> everything is written to the media, by flushing the PC's cache.

```
$>sync
```

Leave the folder since it will disappear when you detach the media from the PC by e.g. going back to the previous folder you were in.

```
$>cd -
```

Remove from PC, and mount onto the RZ/N board. You can read the file system from U-Boot.

```
=>ext4ls mmc 0:1
```

9.2 Bootargs, DTS fdt_chosen Structure

Here is how to modify the system to inform it what physical media contains the root file system. The default is found in the *fdt_chosen* structure of the linux DTS file.

Using bootargs

the *bootargs* environment variable lets you override what is in the *fdt_chosen* structure of the linux DTS. This is great for testing different boot options. Put the string you want to use (replacing any *fdt_chosen*) as the environment variable *bootargs*.

Here are a few examples (see QSG and RZN1 Linux UM).

File system on SD

```
=>setenv bootargs "console=ttyS0,115200 root=/dev/mmcblk0p1 rootfstype=ext4 rw rootwait
ip=192.168.20.40:::::eth0 earlyprintk clk_ignore_unused"
```

File system on QSPI

```
setenv bootargs "console=ttyS0,115200 root=/dev/mtdblock7 init=/init rootwait
ip=192.168.1.40:::::eth0 earlyprintk clk_ignore_unused"
```

You may need the *rfs* environment variable aswell.

```
setenv rfs "root=/dev/mtdblock7 rootfstype=squashfs init=/init"
```

To see the content of the SD card in such case you would need to mount it manually - 'mount /dev/mmcblk0p1 /mnt'.

Changing fdt_chosen permanently

A. Manually

Override *fdt_chosen* of the linux DTS by doing the following at the U-boot prompt.

Start serial flash reader.

```
=>sf probe
```

Read the DTB to RAM.

```
=>sf read 0x8ffe0000 b0000 20000
```

Tell fdt where the DTB is.

```
=>fdt addr 0x8ffe0000
```

List the current value of the device tree chosen *bootargs*.

```
=>fdt print /chosen
```

In U-boot, set *bootargs* to what you want *fdt_chosen* to be. This will override the default `fdt_chosen` settings of the DTS. Here are two examples for the RZ/N1D-EB.

```
=>setenv bootargs "console=ttyS0,115200 earlyprintk debug root=/dev/mmcblk0p1 rootwait
clk_ignore_unused"
=>setenv bootargs "console=ttyS,115200 root=/dev/mmcblk0p1 rootwait init=bin/sh"
```

Finally, you must set *bootargs*.

```
=>fdt chosen
```

If you get something like "Could not set bootargs FDT_ERR_NOSPACE", you need to add more space-characters to the *fdt_chosen* structure. This is explained in 9.2. *Change the default in DTS*.

B. Change the default in DTS

Use this when you know what you want as default bootargs in a release, or to add more characters (spaces) for a longer bootargs. Added space to the *fdt_chosen* structure is make room for long values of *bootargs* should you later need it. If not, *bootargs* is truncated to whatever space is "reserved" in the DTB by the current number of characters (FDT_ERR_NOSPACE error above). This means changing the *fdt_chosen* structure in the linux DTS, recompiling it, and downloading a new DTB.

Open the DTS e.g.

```
../rzn1_linux-4.9_bsp/output/linux-4.9/arch/arm/boot/dts/rzn1d400-db.dts
```

Add space to the end of what you want bootargs to be. This is the line right after
chosen {

Observe the spaces after *rootwait* in our case:

```
bootargs = "console=ttyS0,115200 earlyprintk debug rdinit=/linuxrc root=/dev/mmcblk0p1
rootwait                                                        "; /* ← added space. */
```

Recompile and download the new DTB as described in 6.1, Device Tree.

Details on fdt chosen structure and usage at http://www.denx.de/wiki/view/DULG/UBootCmdFDT

9.3 Boot to File System

Boot to linux and the new file system on the SD. When you get to the prompt, login with

```
$root
```

To see your files, type

```
$>cd /
$ls
```

Your user applications can later be added to folder */usr/bin*.

9.4 Boot Linux from SD

You can boot Linux from the SD instead of loading it from flash (QSPI). Do the following with U-Boot to load the DTB from an SD card. This assumes it is in the */boot* folder.

9.4.1 EXT4

```
=>ext4load mmc 0:1 0x8ffe0000 /boot/uImage_goalctc_1.3.dtb
```

and to load the kernel:

```
=>ext4load mmc 0:1 0x80008000 /boot/uImage_goalctc_1.3.bin
=>setenv bootcmd "sf probe && sf read 0x4000000 d0000 80000 && rzn1_start_cm3 && sleep 4 && bootm
0x80008000 - 0x8ffe0000"
=>boot
```

Or, using bootcmd, autoboot with:

```
=>setenv bootcmd 'ext4load mmc 0:1 0x80008000 uImage-rzn1d400-db.bin; ext4load mmc 0:1 0x80f00000
uImage-rzn1d400-db-no-cm3.dtb; bootm 0x80008000 - 0x80f00000'
```

9.4.2 FAT

If boot section is FAT, replace ext4load with fatload above.

9.5 A Final System

When your application development is completed you may want to run your system standalone, independent from external resources such as an SD, and instead mount the filesystem from a compressed *ramdisk* image stored in e.g. QSPI and loaded into RAM when the system boots. In this case you would instead of issue. The QSG discusses using a *squashfs* file system.

10. Build, Add, Run A7 User Applications

10.1 Cross-compiling

build.sh includes adding a cross-compiler for the A7. You must therefore build Buildroot to cross-compile.

After building Buildroot it is located at on the host PC at
../buildroot/output/host/user/bin/arm-linux-gnueabihf-gcc

When cross compiling, you may run into build problems unless you prepare sufficiently. The most common problems are lack of ownership and permissions to execute certain building scripts.

Permissions

To access or execute files [...*permission denied*] make sure you have access to the build and source tree. For example, permissions are lost when files are copied from Windows. Fix that by changing permissions with e.g.

For example

```
$>sudo chown -R username ~/RZ-N/YCONNECT-IT-RZN
$>sudo chown -R username ~/rzn1-linux-4.9_bsp
```

Some files need to be executable or the commands they contain will not be processed.

```
$>sudo chmod u+x ../../filename
```

Other build tips

If problems, do a Buildroot clean.

```
$>./build.sh buildroot clean
```

10.2 Building the Application

To build your own or other's A7 Linux applications you can use the *build.sh* script.

Go to the directory of the *build.sh* script to set up the host build environment. Don't go to the location of the code you want to build yet.

```
$>cd ~/RZ-N/rzn1_linux-4.9_bsp
$>./build.sh env
```

Follow the directions to set up environment variables to be able to cross compile.

You will see that the terminal font colors changes. This shows you are in "application building mode" with environment variables for example pointing to the arm *gcc* cross-compiler.

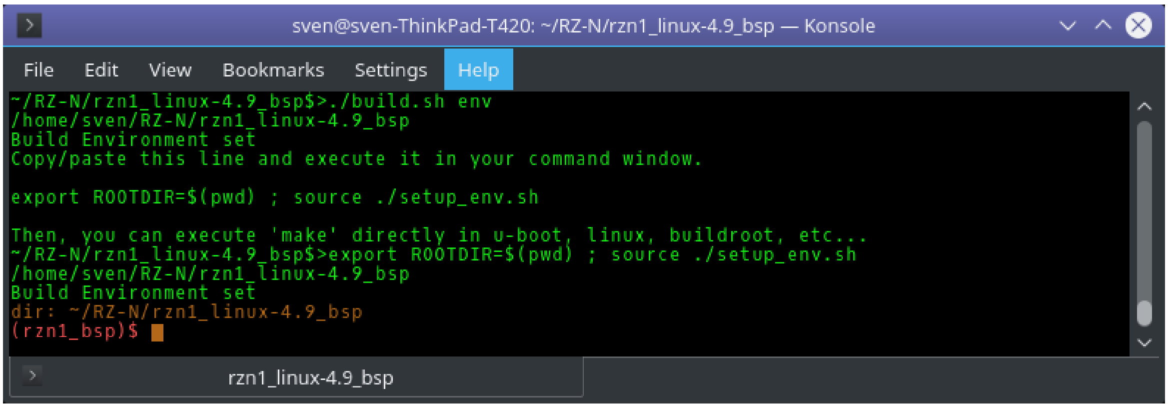


Figure 10. Application building mode. The colors are there to remind you that you are in cross-compile mode, with environment variables set up accordingly.

Now (and only now!) go to the place where you have code you want to build / where your *Makefile* resides.

(Of course, it is pointless to go to a CM3 application directory as these applications are organized for the CM3 environment and don't run under linux.)

Example hello_world

After setting up build environment as shown above, go to the directory where the A7 user application's makefile is, e.g.

```
$>cd hello_world/
$>make
```

If successful, the *file* command should list the output file as a binary executable for the RZ.

```
$>file hello
```

Using Buildroot, add a folder to the tree where additional user files can be placed. Let's use */usr/bin*.

```
$>mkdir -p buildroot-xx.x/output/rootfs_overlay/usr/bin
```

Copy the binary into the tree.

```
$>cp hello buildroot-xx.x/output/rootfs_overlay/usr/bin
```

Now add the user applications to the linux file system's *root_fs* as shown in 8.7.

Example GOAL

Again, go to the *build.sh* directory and set up environment as explained above, then go to the directory where the *Makefile* is for the A7 project you want to build.

```
$>cd ~/RZ-N/YCONNECT-IT-RZN_V1.1/Software/GOAL/goal/projects/00410_goal/ctc_ac/gcc
```

Build the code.

```
$>make
```

Copy the binary into the tree (after creating the destination folder of the tree as in the hello example above).

```
$>cp ./build/goal/goal_rzn_a7_demo_board.bin buildroot-xx.x/output/rootfs_overlay/usr/bin
```

Auto-Start an A7 Application

Use Linux's *inittab* functionality to do this. Information how to do this is in the Goal Profinet or Powerlink QSG section Auto start the user application

See e.g. online documentation. Buildroot documentation should also explain this.

10.3 Upload Files to Deployed System

Target

See 8.7 on how to add the *lrzsz* executable to the target file system. Once you have this available on the target, issue the following command to receive a file at target (upload from host).

```
$ rz -y
(or $ rz -Z)
```

[Will lock-up and wait for host.]

Host PC

Install *lrzsz* with your favorite package manager, then go to the folder containing the file you want to transfer.

```
$>cd hello_world
>sz -b hello > /dev/ttyUSB2 < /dev/ttyUSB2
```

Another example; from a Goal project's make directory.

```
$>sz -b build/rzn_a7_demo_board/goal_rzn_a7_demo_board__ctc_1.3_pure.bin  > /dev/ttyUSB2 <
/dev/ttyUSB2
```

11. Cortex M3 Application Example

See section 7, *Cortex M3 Applications* how to run and debug.

11.1 Port Gmbh Based Demos

11.1.1 TCP Server

1. Open the IAR-EW workspace

```
..\RZ-N\YCONNECT-IT-RZN_Vx\Software\GOAL\goal\projects\00410_goal\tcp_server\iar\...
\rzn_demo_board_eb\rzn_demo_board
```

2. Check the main board switches as set in the doc. RZ_N1D_DB_Board_V800_Setup_Notes. Switch W-1 needs be OFF (ARM Coresight) to debug, and SW2-6 ON, SW2-7 OFF, SW2-8 ON. (Below code tested with all W-x switches OFF - away from white bar.)
3. Connect the RZ/N Expansion board's Ethernet jack J23 to the PC.
4. Set the PC IP address to 192.168.1.X (e.g. x = 50), that is, on the same sub-domain as the board's IP-address.
5. The board's IP-address is SERVER_IP_ADDRESS. This is the same as MAIN_APPL_IP found in file *goal_appl.c* of the server code. 192.168.1.3 by default in the code.
6. To run and debug, follow RZ-N1D-System-Setup-Tutorial section 7.3.2 "Debug CM3 with Application Running on A7".

12. Appendix - DFU Utility Information

Firstly, see sections *dfu-util for Linux* and dfu-util for Microsoft Windows in the RZN1-U-Boot-User-Manual.

12.1 Problems

If you are installing U-Boot, try SW4 instead of SW3.

See below for Linux and Windows specifics.

12.1.1 Windows

You can run DFU from Windows instead of a Linux host. Download the DFU utility from http://dfu-util.sourceforge.net/releases/

A usage example:

```
>dfu-util-static.exe -a 'sf_kernel' -D 'C:\Workspace\RZN\YCONNECT-IT-RZN_V1.3.1\Software\U-Boot-and-Linux\kernel\binaries\uImage'
```

- Make sure you see the DFU driver in Device Mgr. when you enter command *dfu* into the RZ/N serial console.

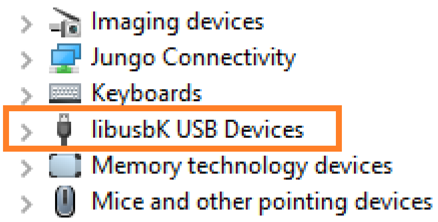


Figure 11. The *libusbK USB device* should be visible in Device Mgr to do a DFU download.

- Use Windows Powershell. The standard command prompt may not find the DFU USB device.

12.1.2 Linux

- You may need to change file properties of the executable, in this case *dfu-util*.

```
$>chmod +x ./executable_file
```

- Unless you are sudo, you may need to change ownership of the executable, in this case *dfu-util*.

```
$>chown username ./executable_file
```

- If you are usng a virtual linux host, check that e.g. *Linux usbf Renesas* or similar is enumerated to the virtual PC instead of Windows.

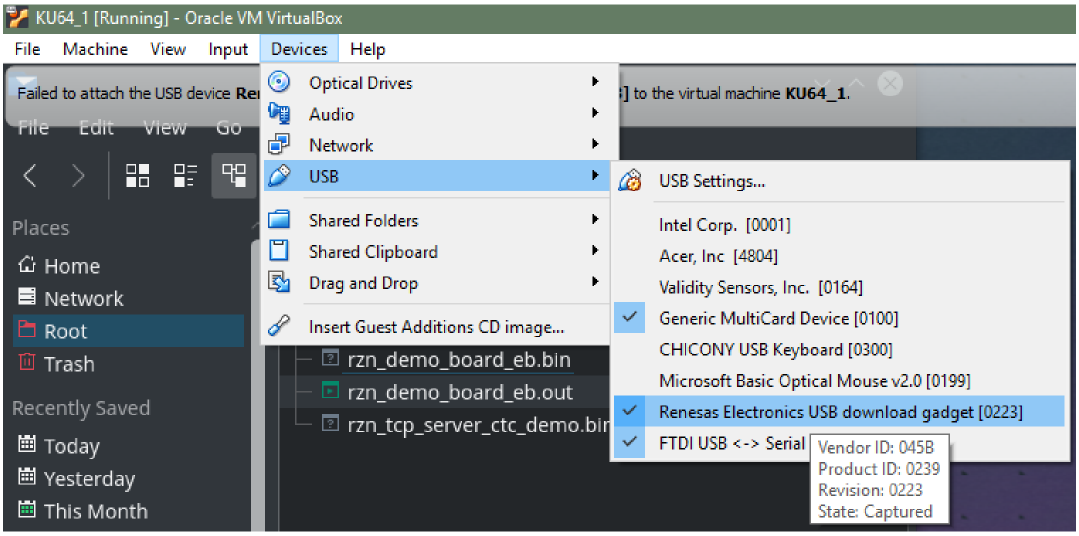


Figure 12. A virtual linux host needs to have access the hardware attached to the PC.

Website and Support

Renesas Electronics Website

http://www.renesas.com/
RZ/N1-1 homepage
RZN1 DVD

Inquiries

http://www.renesas.com/contact/

Revision History

Rev.	Date	Description: Page	Description: Summary
1.00	Jan 2, 2018		- Revised wording of chapter 9. (Implement a Custom System with Buildroot.) - Expanded 5.3.1 bootcmd table. - Moved *sf_probe* command in 7.3.2. to right before *sf_read* (more logical).
1.01	Jan 11, 2018		Re-export to PDF as there were corrupted parts in section 9.
1.02	Mar 6, 2018		Added 11. General revision.
1.03	Apr 3, ‘18		Added Cross-compile info.
1.04	May 14, ‘18		- Added sections on how to download target files to an alrady deployed system (“Example; adding target download ability to Buildroot”) under 8.7 and section 10.3. - Merged chapters 8 and 9 into one (now 8).
1.05	Jun 1, ‘18		- Updated language slightly. Links updated.
1.06	Jun 7, ‘18		- Added Figure 13. SFTP for connecting to get the RZ-N DVD content and more. - Fixed broken internal doc. references.
1.07	Jun 27, ’18.		- Changed SFTP info. - Added comment on needed host packages when using *rzn1_linux-4.9_bsp*.
1.08	Jul 3, ‘18		- Linux Virtual machine tips added. - Clarifications in 4.2 and 5. - Added initial use of *rzn1_defconfig* to 8.3.
1.09	Aug 1, ‘18		- Added 1.1 Power. - Enhanced chapter 2 for getting the rzn1_linux-4.9_bsp package. - Added 6.2.3 and 9.4 (boot from SD). - Changed chapter hierarchy from chapter 8 and forward.
1.10	Aug 14, ‘18		- 8.1; switched to rzn1_linux-4.9-bsp**2.** Added 8.3 Quick Summary and how to update kernel with latest in 8.4.
1.11	Oct 26, 18		- Changed location of DTB in DDR to be same as latest DVD documentation: From 0x80FE0000 to 0x8FFE0000. - Removed webserver CM3 example. It was old SICS version. Had to use a tool makefsdata to create web server content. Use A7-side on N1D or S to do webserving. - Added figure showing libusbK device to Appendix. - Updated DVD links etc in References.
1.12	Oct 30, ‘18		- Added info on using QSPI as root file system (squashfs) in section 9. - How to access file system from U-Boot.
1.13	Nov 13, ‘18		- Changed Inotroduction. - Reworded section *DTS fdt_chosen Structure* + *U-Boot*. - Added U-Boot sections *SD/MMC* and *Boot to Linux*.
1.14	Jan 15, 2018		- Amplified section *Boot Linux from SD*. -

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

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

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

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

(Rev.4.0-1 November 2017)

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338