

Yocto DevKit for SBC-RZN1D

Documentation

Rev006 / 08.10.2021

© Copyright 2018 **emtrion GmbH**

All rights reserved. This documentation may not be photocopied or recorded on any electronic media without written approval. The information contained in this documentation is subject to change without prior notice. We assume no liability for erroneous information or its consequences. Trademarks used from other companies refer exclusively to the products of those companies.

Revision: **006 / 08.10.2021**

Rev	Date/Signature	Changes
001	Wi/04.06.2018	First release
002	Wi/06.08.2018	Minor changes in the bitbake process
003	Wi/23.04.2019	Rebranding changes
004	MI/08.02.2021	Yocto: changed from morty to dunfell, supporting Init-Manager systemd and sysvinit, changing layer structure, changed U-Boot environment for cm3 support, removing VM due to license issue, removing eclipse
005	MI/23.04.2021	Extend the emtrion meta-layer supporting Preempt-RT, supporting eeprom, emPURS: check for image file before deleting the flash
006	MI/08.10.2021	Changed Uboot command sequence in setting of the serverip after executing of dhcp in any case

Inhaltsverzeichnis

1	INTRODUCTION	4
2	TERMS AND DEFINITIONS	5
3	SCOPE OF THE DELIVERY	7
4	HOST DEVELOPMENT MACHINE	8
4.1	CHOICE OF THE LINUX DISTRIBUTION.....	8
4.2	FURTHER REQUIREMENTS.....	8
4.2.1	<i>Predefined directories</i>	8
4.2.2	<i>Getting the delivery</i>	9
4.2.3	<i>Serial port</i>	9
4.2.4	<i>NFS server</i>	9
4.2.5	<i>TFTP server</i>	9
4.2.6	<i>Serial terminal</i>	9
4.2.7	<i>Yocto Layer</i>	9
4.2.8	<i>Installing the prebuilt target RFS for booting from NFS Server</i>	9
4.2.9	<i>Installing emtrion's update mechanism emPURS</i>	9
4.2.10	<i>Miscellaneous</i>	9
4.3	DEVICE START UP.....	10
4.4	DEVICE NETWORK SETUP.....	11
4.5	PREBUILT IMAGES AND INSTALLATIONS	13
4.5.1	<i>Target Root Filesystems</i>	13
4.5.2	<i>SDK Root Filesystems</i>	14
4.5.3	<i>SDK-Installer</i>	14
5	THE META-LAYER FOR SBC-RZN1D	15
5.1	PREPARING THE ENVIRONMENT WITH OR WITHOUT RT SUPPORT.....	15
5.2	CREATING THE IMAGES	16
5.2.1	<i>core-image-purs</i>	16
5.2.2	<i>emtrion-image-sbc-rzn1d(-systemd)</i>	16
5.2.3	<i>emtrion-image-sbc-rzn1d(-systemd)-sdk</i>	16
5.2.4	<i>SDK installer</i>	17
5.2.5	<i>Miscellaneous</i>	17
5.3	OUTPUT FILES	17
5.3.1	<i>Root Filesystem</i>	18
5.3.2	<i>boot directory</i>	18
6	U-BOOT BOOTLOADER.....	19
6.1	BASIC U-BOOT OPERATION	19
6.2	THE UBOOT_SCRIPT.....	19
6.2.1	<i>Implemented commands</i>	20
6.3	USING U-BOOT TO CHANGE BOOT DEVICE OR UPDATE PARTS OF THE SYSTEM	21
6.3.1	<i>Boot setup and updating the RFS</i>	21
6.3.2	<i>Updating the RFS (using NFS)</i>	22
6.3.3	<i>Updating of U-Boot bootloader (using TFTP)</i>	22
6.3.4	<i>Updating of U-Boot SPL (using TFTP)</i>	23

6.3.5	Booting	23
6.3.6	Boot from on-board flash	23
6.3.7	Boot via network using a NFS share.....	24
7	SDK.....	25
7.1	INSTALLING THE SDK.....	25
7.1.1	Setting up the SDK environment	25
7.1.2	SDK Root Filesystem	26
8	FURTHER INFORMATION	26
8.1	ONLINE RESOURCES.....	26
8.2	WE SUPPORT YOU	26

1 Introduction

Emtrion produces and offers various boards and modules which are available at <https://support.emtrion.de/en/home.html>. One of the exciting products in Emtrion`s product range is the single board named SBC-RZN1D. This manual provides instructions and pointers for efficient use of Yocto Project development with the hardware.

The SBC-RZN1D is based on the processor RZN1D from Renesas. The processor consists of a Dual Cortex-A7(500MHz) and a Cortex-M3(125MHz). The processor implements some special features for using in industrial communication such as PROFINET, EtherCAT and network switch.

The Yocto version for the product is Yocto dunfell (3.1.2). Due to Renesas didn't just yet provide a newer Yocto layer for the RZN1D than Yocto Rocko, emtrion does now provide one for Yocto dunfell.

The BSP contains a Linux kernel based on version 4.9.0 and U-Boot Version 2017.01 provided by Renesas. Both are optimized and adapted by emtrion to perfectly fit the SBC-RZN1D.

From **rev 005** at this document, the BSP also supports Preempt-RT.

The important version information for the various tools and packages are listed below.

Item	Version information
linux distribution Host	Debian Buster
meta-emtrion-rzn1	tag vSBC-RZN1D -1.1 based on branch dunfell
linux target	v4.9.0, rzn1-stable, tag vSBC-RZN1D-1.1 based on branch sbc-rzn1d
linux-rt target	V4.9.201-rt134, tag vSBC-RZN1D-rt-1.0 based on branch sbc-rzn1d-rt
u-boot target	2017.01 sbc-rzn1d v01.000, rzn1-stable
Yocto	dunfell v3.1.2

bitbake	v1.46.0
gcc	v9.3.0
openssh	v8.2
busybox	v1.31.1
initscripts	v1.0

This manual describes the contents of the developer kit and the general information as well as instructions for the user.

It is assumed that users of emtrion Linux developer kits are already familiar with U-boot, Linux, Yocto and creating and debugging applications. General Linux and programming knowledge are out of the scope of this document. emtrion is happy to assist you in acquiring this knowledge. If you are interested in training courses or getting support, please contact the emtrion sales department.

The important resources that can be accessed for further in-depth knowledge are as follows:

1. <https://docs.yoctoproject.org/>
2. <https://docs.yoctoproject.org/releases.html#dunfell-release-series>
3. <https://www.yoctoproject.org/docs/3.1.2/ref-manual/ref-manual.html#ref-manual-system-requirements>
4. [Yocto Project Application Development and the Extensible Software Development Kit \(eSDK\)](#)
5. Yocto repos.: <https://git.yoctoproject.org/cgi/cgi.cgi/poky/tree/?h=dunfell>
6. Kernel repos.: https://github.com/renesas-rz/rzn1_linux/tree/rzn1-stable
7. U-boot repos.: https://github.com/renesas-rz/rzn1_u-boot/tree/rzn1-stable

2 Terms and Definitions

Term	Definition
Target	Board SBC-RZN1D
Host	Workstation, Developer PC
Toolchain	Compiler, Linker, etc.
RFS	Root file system, contains the basic operating system
Console	Text terminal interface for Linux
NFS	Network File System, which can share directories over network
NFS_SHARE	Location that is exported by the NFS for the purpose of updating and booting by using NFS
U-Boot	Bootloader, hardware initialization, updating images, starting OS
YP	Yocto Project

INST_DIR	Directory where Yocto and the meta-layers are installed
MACHINE	Specifies the target device for which the image is built. The variable is named sbc-rzn1d for this kit
BUILD_DIR	Machine dependent build directory
BSP	Board Support Package
SDK	Software Development Kit

3 Scope of the delivery

Other than before, this kit doesn't come with a prepared Virtual Machine. License issues are the reasons.

Subsequently, emtrion provides the corresponding software and prebuilt binaries via cloud.

Buyers will get a cloud link. This link points to a directory which includes the gz-Archiv **sbc-rzn1d_dunfell.tar.gz**, including a defined directory structure as below. The gz-Archiv contains all the images with and none RT support.

- meta-layer
 - contains the Yocto layer **meta-emtrion-rzn1** for producing the various images
- RFS
 - contains the prebuilt RFS images with systemd and sysvinit support
 - emtrion-image-sbc-rzn1d.tar.bz2
 - emtrion-image-sbc-rzn1d-sdk.tar.bz2
 - emtrion-image-sbc-rzn1d-systemd.tar.bz2
 - emtrion-image-sbc-rzn1d-systemd-sdk.tar.bz2
 - emtrion-image-sbc-rzn1d-rt.tar.bz2
 - emtrion-image-sbc-rzn1d-sdk-rt.tar.bz2
 - emtrion-image-sbc-rzn1d-systemd-rt.tar.bz2
 - emtrion-image-sbc-rzn1d-systemd-sdk-rt.tar.bz2
- SDKs
 - contains the SDK installer to make application development useable outside the yocto build system. You will find the installers in the corresponding directories, each for none-RT and RT.
 - **none-RT**
 - poky-glibc-x86_64-emtrion-image-sbc-rzn1d-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh
 - poky-glibc-x86_64-emtrion-image-sbc-rzn1d-systemd-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh
 - **RT**
 - poky-glibc-x86_64-emtrion-image-sbc-rzn1d-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh
 - poky-glibc-x86_64-emtrion-image-sbc-rzn1d-systemd-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh
- restore
 - contains the files of emtrion's update mechanism emPURS for updating the target's RFS

- zImage
- sbc-rzn1d-rdsk.dtb
- u-boot_script
- emPURS_plat
- initramfs-sbc-rzn1d.cpio.gz
- u-boot
 - contains the prebuilt binaries U-Boot and SPL
 - u-boot.img
 - spl-rel.spkg

4 Host Development Machine

Due to not any prepared VM is available, you have to setup your own linux distribution before working with the kit. However, setting up the linux distribution is not so much work. Below is a short guideline.

4.1 Choice of the linux distribution

The Yocto Project supports several linux distributions and what are the requirements. For Yocto dunfell, please look at <https://www.yoctoproject.org/docs/3.1.2/ref-manual/ref-manual.html#ref-manual-system-requirements> to meet the requirements. Please note, the linux distribution **Debian Buster** is used by this kit and so we prefer this for your system too.

4.2 Further requirements

4.2.1 Predefined directories

This document references to some predefined directories. Following this document we recommend you to provide this predefined directories.

Placeholder	Path	Remark
HOME_DIR	/home/hico	home directory of user hico(to replace it with the corresponding home directory on your system)
DWL_DIR	<HOME_DIR>/Downloads	the users download directory
NFS_SHARE	<HOME_DIR>/nfs	location exported by the NFS
NFS_ROOTFS	<NFS_SHARE>/rootfs	nfs share for booting the RFS by using NFS
NFS_RESTORE	<NFS_SHARE>/restore	location to put the emPURS files for updating the onboard flashes
SDK_DIR	<NFS_SHARE>/sdk	install directory for the SDK
INST_DIR	<HOME_DIR>/yocto/build	install directory where Yocto and all the meta-layers will be stored to
BUILD_DIR	<INST_DIR>/builddir	build directory of the build system.
TFTP_DIR	/srv/tftp	used by the TFTP server for exporting files

4.2.2 Getting the delivery

Download the gz-Archiv **sbc-rzn1d_dunfell.tar.gz** from the cloud and save it to the folder **<DWL_DIR>**. Then decompress it in this folder by

```
tar xf sbc-rzn1d_dunfell.tar.gz
```

4.2.3 Serial port

A USB serial converter appearing at **/dev/ttyUSBn**

4.2.4 NFS server

Setting up a NFS server and export the **<NFS_SHARE>**. This can be different from the linux distribution. Be sure to make this work correct, please inform yourself how this work has to be done.

The entry in **/etc/exports** look like as follows

```
<NFS_SHARE> *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

4.2.5 TFTP server

We assume the tftp directory at **<TFTP_DIR>**

4.2.6 Serial terminal

We assume the terminal program **picocom** for connecting to the target

4.2.7 Yocto Layer

Create the directory **<INST_DIR>** and install the Yocto Layer **meta-emtrion-rzn1** from **<DWL_DIR>** to it.

4.2.8 Installing the prebuilt target RFS for booting from NFS Server

Create the directory **<NFS_ROOTFS>** and decompress the interested target RFS with or none RT, depending on the image name, **emtrion-image-sbc-rzn1d(-rt).tar.bz2**, from **<DWL_DIR>** to it.

```
tar xf <DWL_DIR>/RFS/emtrion-image-sbc-rzn1d(-rt).tar.bz2 -C <NFS_ROOTFS>
```

4.2.9 Installing emtrion's update mechanism emPURS

Create the directory **<NFS_RESTORE>** and copy the emPURS files from **<DWL_DIR>** to it. Then create the subdirectory "images" in **<NFS_RESTORE>**.

```
cp <DWL_DIR>/restore/* <NFS_RESTORE>/
```

```
mkdir <NFS_RESTORE>/images
```

4.2.10 Miscellaneous

Previous delivered boards that containing an old U-Boot version has to be updated first.

To find out the version of U-Boot, please power on the SBC-RZN1D and stop booting by pressing any key. Then enter the command

- **printenv ver**

If the version does not start with the string “**U-Boot 2017.01 sbc-rzn1d v...**” it is an old one. In this case you have to update the SPL, U-Boot and the target’s RFS first. This is described here 6.3.1.1

4.3 Device Start Up

Connect the developer kit to the serial port attached to your system and to your network. Open a console in the linux and open a serial terminal by entering:

```
sudo picocom -b 115200 /dev/ttySx
```

ttySx has to be replaced with the device assigned to the connected serial port e.g. ttyS1. In the case of using an USB serial adapter replace it by the corresponding **ttyUSBn**.

You may now power on the developer kit or enter the command `res`. You should see it booting U-Boot and Linux from Flash.

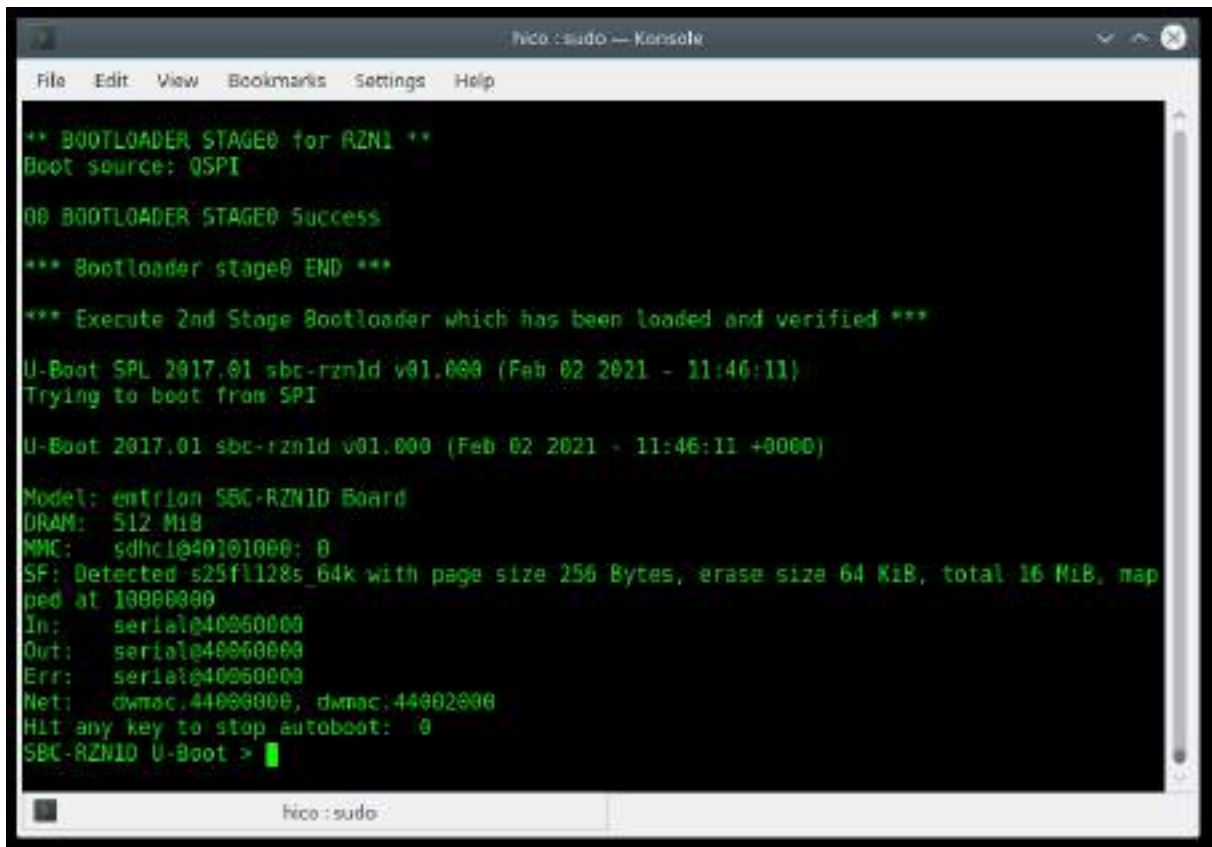


Figure 1: Serial terminal showing U-Boot prompt

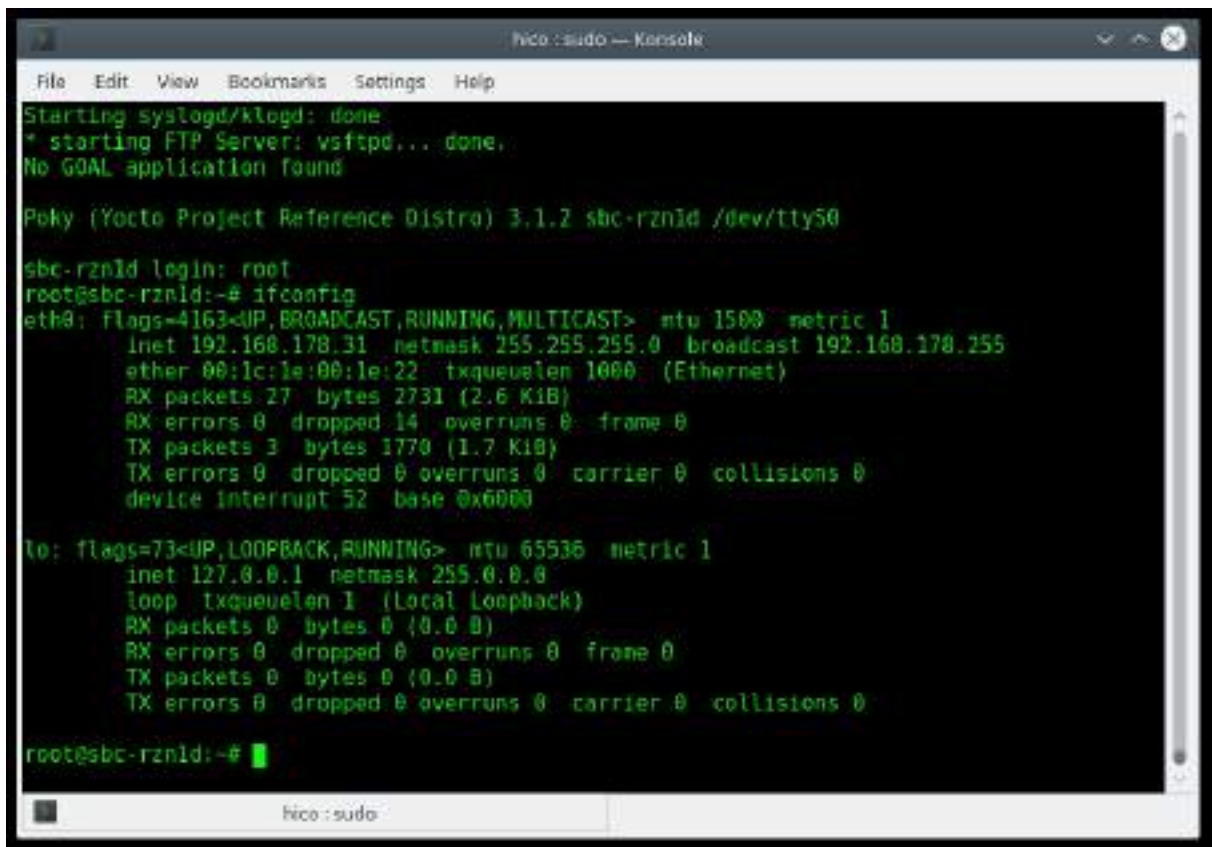
After the developer kit booted you are prompted for login:

- **user: root**
- **password: no password set**

4.4 Device Network Setup

Per default the developer kit is setup to use a DHCP server. This is configurable by a bootloader environment variable "ip-method". This variable can have the values "dhcp" or "static".

You can check if there is a valid IP address with the command "ifconfig" or "ip addr show eth0".



```
hico: sudo - Konsole
File Edit View Bookmarks Settings Help
Starting syslogd/klogd: done
* starting FTP Server: vsftpd... done.
No GOAL application found

Poky (Yocto Project Reference Distro) 3.1.2 sbc-rzn1d /dev/tty50
sbc-rzn1d login: root
root@sbc-rzn1d:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 metric 1
    inet 192.168.178.31 netmask 255.255.255.0 broadcast 192.168.178.255
    ether 00:1c:1e:00:1e:22 txqueuelen 1000 (Ethernet)
    RX packets 27 bytes 2731 (2.6 KiB)
    RX errors 0 dropped 14 overruns 0 frame 0
    TX packets 3 bytes 1770 (1.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 52 base 0x6000

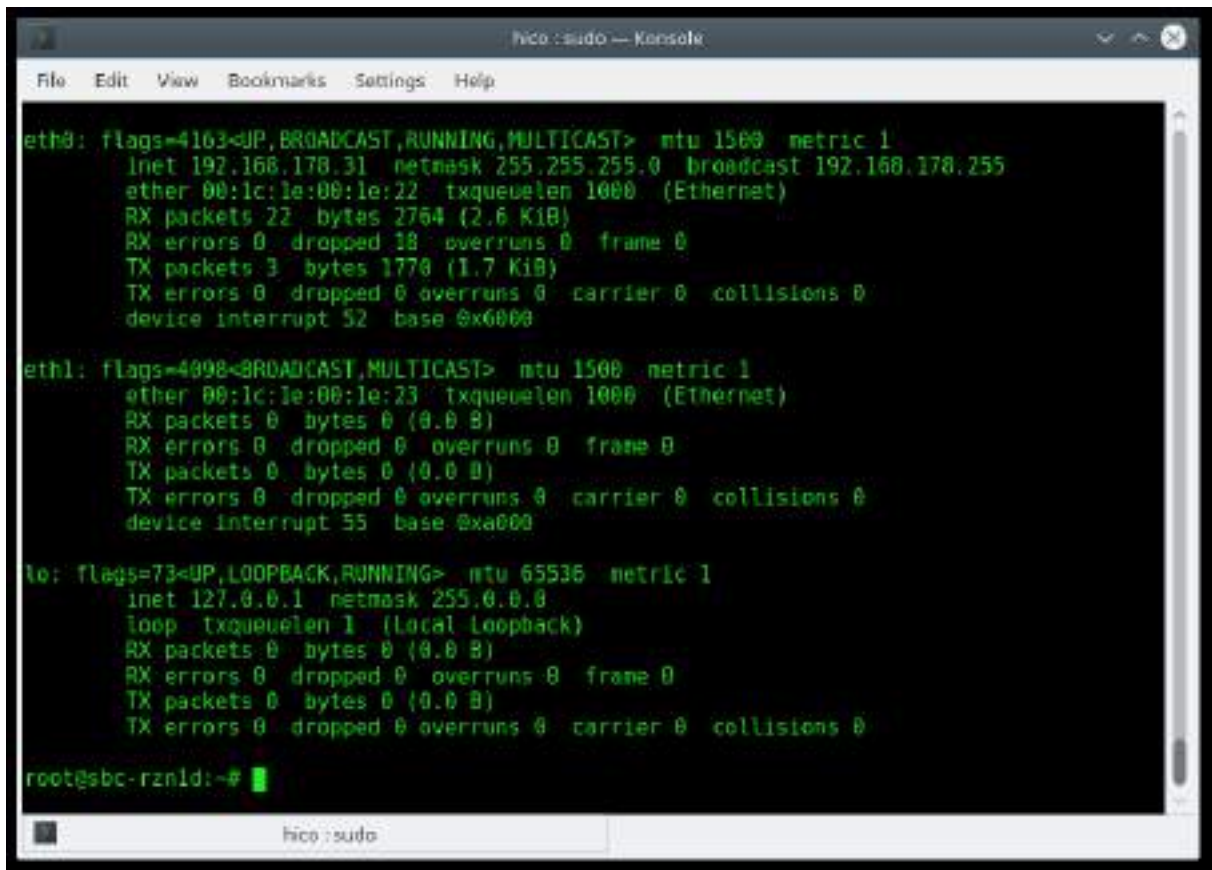
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 metric 1
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@sbc-rzn1d:~#
```

Figure 2: ifconfig output

If the setup is not correct you have to do it manually. Please check the description of the bootloader configuration on how to set up the variable "ip-method".

The device is supporting a second network interface eth1. You can show it by the command ifconfig -a.



```
Nico: sudo — Konsole
File Edit View Bookmarks Settings Help

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 metric 1
  inet 192.168.178.31 netmask 255.255.255.0 broadcast 192.168.178.255
  ether 00:1c:1e:00:1e:22 txqueuelen 1000 (Ethernet)
  RX packets 22 bytes 2754 (2.6 KiB)
  RX errors 0 dropped 18 overruns 0 frame 0
  TX packets 3 bytes 1770 (1.7 KiB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  device interrupt 52 base 0x6000

eth1: flags=4098<BROADCAST,MULTICAST> mtu 1500 metric 1
  ether 00:1c:1e:00:1e:23 txqueuelen 1000 (Ethernet)
  RX packets 0 bytes 0 (0.0 B)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 0 bytes 0 (0.0 B)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  device interrupt 55 base 0xa000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 metric 1
  inet 127.0.0.1 netmask 255.0.0.0
  loop txqueuelen 1 (Local Loopback)
  RX packets 0 bytes 0 (0.0 B)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 0 bytes 0 (0.0 B)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@sbc-rzn1d:~#
```

Figure 3: ifconfig -a output

If the interface **eth1** connected to a network providing a dhcp, you can request an IP address by the command `dhclient eth1`.

```

Nico: sudo - Konsole
File Edit View Bookmarks Settings Help
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 metric 1
  inet 192.168.178.31 netmask 255.255.255.0 broadcast 192.168.178.255
  ether 00:1c:1e:00:1e:22 txqueuelen 1000 (Ethernet)
  RX packets 570 bytes 39359 (38.4 KiB)
  RX errors 0 dropped 486 overruns 0 frame 0
  TX packets 6 bytes 1908 (1.8 KiB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  device interrupt 52 base 0x6000

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 metric 1
  inet 192.168.178.32 netmask 255.255.255.0 broadcast 192.168.178.255
  ether 00:1c:1e:00:1e:23 txqueuelen 1000 (Ethernet)
  RX packets 11 bytes 1720 (1.6 KiB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 2 bytes 684 (684.0 B)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  device interrupt 55 base 0xa000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 metric 1
  inet 127.0.0.1 netmask 255.0.0.0
  loop txqueuelen 1 (Local Loopback)
  RX packets 0 bytes 0 (0.0 B)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 0 bytes 0 (0.0 B)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@sbc-rzn1d:~#

```

Figure 4: ifconfig -a after dhclient eth1

4.5 Prebuilt images and installations

As mentioned before, the kit is provided by various prebuilt binaries for downloading by a cloud link.

The various binaries are assigned at specific subdirectories.

There are RFS images with sysvinit and systemd support as well with SDK extensions, each for none RT and RT. All this information is reflected by the name of the corresponding image. Other than in case of systemd, sysvinit has not be inserted in the image name.

In order to create a new root filesystem yourself, please follow the steps described in chapter 5.

4.5.1 Target Root Filesystems

The target root filesystems for the SBC-RZN1D are located in RFS.

RFS

- └─ emtrion-image-sbc-rzn1d-systemd.tar.bz2
- └─ emtrion-image-sbc-rzn1d-systemd-rt.tar.bz2

- |— emtrion-image-sbc-rzn1d.tar.bz2
- └— emtrion-image-sbc-rzn1d-rt.tar.bz2

For booting from NFS you can decompress the interested RFS image to **<NFS_ROOTFS>**

tar xf <DWL_DIR>/RFS/emtrion-image-sbc-rzn1d(-sdk)(-rt).tar.bz2 -C <NFS_ROOTFS>

4.5.2 SDK Root Filesystems

The SDK root filesystems are located in RFS, too.

RFS

- |— emtrion-image-sbc-rzn1d-systemd-sdk(-rt).tar.bz2
- └— emtrion-image-sbc-rzn1d-sdk(-rt).tar.bz2

The extracted SDK Root Filesystem is part of the SDK and for development purpose and not suitable for normal use. You need this root filesystem during the development for your applications. How you can boot the installed root filesystem using NFS is described in the paragraph *SDK Root Filesystem* chapter 7.1.2.6.3.7

4.5.3 SDK-Installer

Like the target root filesystems, there are SDK installers each for Preempt-RT and none for the sysvinit and the systemd RFS. They are located in the corresponding subdirectories.

SDKs

none-RT

- |— poky-glibc-x86_64-emtrion-image-sbc-rzn1d-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh
- └— poky-glibc-x86_64-emtrion-image-sbc-rzn1d-systemd-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh

RT

- |— poky-glibc-x86_64-emtrion-image-sbc-rzn1d-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh
- └— poky-glibc-x86_64-emtrion-image-sbc-rzn1d-systemd-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh

How are using the SDK installer is described in chapter 7.1.

5 The meta-layer for SBC-RZN1D

NOTE: If you do not want to create your own root filesystems, you can skip this chapter.

The meta-layer **meta-emtrion-rzn1** is based on the kernel recipe and U-Boot recipe from Renesas for the RZ/N1 SoCs with some additions to adapt emtrion's board SBC-RZN1D.

The meta-layer supports **none RT** and **RT** images and is located in

meta-layer

└─ meta-emtrion-rzn1

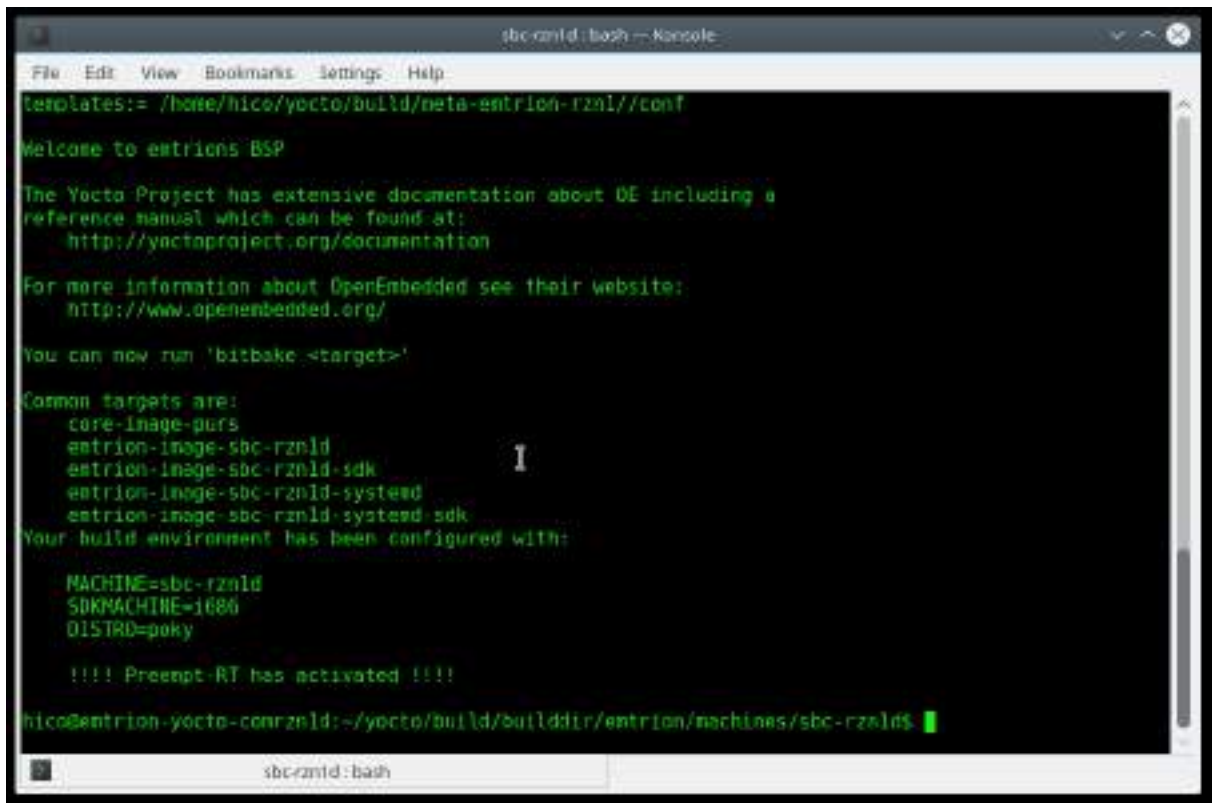
If you have not already installed the meta-layer to <INST_DIR>, please do it now.

5.1 Preparing the environment with or without RT support

Before creating an image, you will have to prepare the build environment accordingly. You can define whether are creating images **with Preempt-RT** or **none**. First go to the <INST_DIR>/**meta-emtrion-rzn1** directory and execute the setup script:

- ❖ RT
 - **RT=y source setup_environment**
- ❖ none RT
 - **source setup_environment**

This will download all the needed yocto layers and prepares the environment. At the end of the setup process the build environment is prompted with some information.



```
sbcrzn1d: bash — Konsole
File Edit View Bookmarks Settings Help
templates:= /home/hico/yocto/build/meta-emtrion-rzn1d/conf
Welcome to emtrions BSP
The Yocto Project has extensive documentation about OE including a
reference manual which can be found at:
  http://yoctoproject.org/documentation
For more information about OpenEmbedded see their website:
  http://www.openembedded.org/
You can now run 'bitbake <target>'
Common targets are:
  core-image-purs
  emtrion-image-sbc-rzn1d
  emtrion-image-sbc-rzn1d-sdk
  emtrion-image-sbc-rzn1d-systemd
  emtrion-image-sbc-rzn1d-systemd-sdk
Your build environment has been configured with:
  MACHINE=sbc-rzn1d
  SDKMACHINE=i686
  DISTRO=poky
  !!!! Preempt-RT has activated !!!!
hico@emtrion-yocto-comrzn1d:~/yocto/build/builddir/emtrion/machines/sbc-rzn1d$
```

In addition to the supported MACHINE you will see the target images can be created and RT is activated or not. In the above case we setup the environment with RT=y.

5.2 Creating the images

Now it's time to start building images for emtrion's board SBC-RZN1D.

The build process is done in four steps and is identical in case of RT or not. The first step is common for the sysvinit and systemd images. The other three steps are dependent on sysvinit or systemd.

5.2.1 core-image-purs

The first step creates an initramfs which is part of emPURS (emtrion Production, Update and Recovery System):

```
bitbake core-image-purs
```

5.2.2 emtrion-image-sbc-rzn1d(-systemd)

The second one creates the target root filesystem image:

```
bitbake emtrion-image-sbc-rzn1d(-systemd)
```

5.2.3 emtrion-image-sbc-rzn1d(-systemd)-sdk

The third creates the SDK root filesystem image:

```
bitbake emtrion-image-sbc-rzn1d(-systemd)-sdk
```


5.2.4 SDK installer

After you have created the SDK root filesystem image, you can now generate the SDK installer:

```
bitbake emtrion-image-sbc-rzn1d(-systemd)-sdk -c populate_sdk
```

5.2.5 Miscellaneous

Due to the meta-layer contains recipes for the kernel and U-Boot, you have the possibility to create these binaries separately. But this is not necessary because kernel and U-Boot are automatically created and included in the RFS by step 2 as well by step 3.

- U-Boot and SPL generation
 - **bitbake u-boot-rzn1**
- kernel generation
 - **bitbake linux-rzn1(-rt)**

5.3 Output files

The images or binaries produced by the previous bitbake processes are installed in the two central directories of the build system.

```
<BUILD_DIR>/emtrion/machines/sbc-rzn1d/tmp/deploy/images/sbc-rzn1d
```

and

```
<BUILD_DIR>/emtrion/machines/sbc-rzn1d/tmp/deploy/sdk
```

The names of the RFS images, depend on the RT is activated or not. Following the outputs are listed.

Images	description
u-boot.img	U-Boot
spl-rel.spkg	SPL
zImage	Linux Kernel
sbc-rzn1d.dtb	Device tree RFS
sbc-rzn1d-rdsk.dtb	Device tree for emPURS
initramfs-sbc-rzn1d.cpio.gz	initramfs
emtrion-image-sbc-rzn1d(-rt).tar.bz2	RFS
emtrion-image-sbc-rzn1d-sdk(-rt).tar.bz2	RFS for SDK
emtrion-image-sbc-rzn1d-systemd(-rt).tar.bz2	RFS(systemd+)
emtrion-image-sbc-rzn1d-systemd-sdk(-rt).tar.bz2	RFS for SDK
poky-glibc-x86_64-emtrion-image-sbc-rzn1d-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh	SDK installer
poky-glibc-x86_64-emtrion-image-sbc-rzn1d-systemd-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh	SDK installer

5.3.1 Root Filesystem

As shown in the list above, the output of the RFS is a tar.bz2 archive. You can decompress it using the tar command. For test purpose we recommend to decompress the archive to the **<NFS_ROOTFS>**. Be sure the **<NFS_ROOTFS>** is empty before decompressing.

From the prompt at the build terminal **<BUILD_DIR>/emtrion/machines/sbc-rzn1d**

call **sudo tar xf ./tmp/deploy/images/sbc-rzn1d/<rootfs-name>.tar.bz2 -C <NFS_ROOTFS>**

Don't forget "sudo" otherwise the kernel won't be able to modify the files during start up of the system.

5.3.2 boot directory

The directory structure of the root file system includes a directory **/boot**. Among the located files there is a file **uboot_script**.

This text file implements some U-Boot command sequences. You can use it for the purpose of updating and booting via network, like kernel, U-Boot and RFS.

However, the environment of the U-Boot has to be set up before. This is discussed in detail in chapter 6.3.1.

6 U-Boot Bootloader

The basic task of U-Boot is to load the operating system from bulk memory into RAM and then start the kernel. You can also use it to initiate an update of the RFS and of U-Boot itself. Furthermore you can configure the medium the operating system should be booted from, for example eMMC, NFS or a serial terminal.

6.1 Basic U-Boot operation

To work with U-Boot, first use a terminal program like picocom to connect to the serial line of the board. As soon as the U-Boot prompt appears in the terminal, U-Boot is ready to receive commands. The general U-Boot documentation can be found here: <http://www.denx.de/wiki/U-Boot/Documentation>

U-Boot has a set of environment variables which are used to store information needed for booting the operating system. Variables can contain information such as IP addresses, but they can also contain a whole script of actions to perform sequentially. The following commands explain the basic handling of environment variables:

U-Boot command	Explanation
printenv [variable]	This shows the value of the specified variable. If no variable is specified, the whole environment is shown.
setenv [variable] [value]	Set a variable to a specific value. If no value is specified, the variable gets deleted.
saveenv	Make your changes permanent, so they remain after power off or reboot.

6.2 The uboot_script

The uboot_script provides some helpful command sequences for the purpose of updating and booting. The script is necessary for booting and is part of the RFS. It is located in the directory **/boot**.

At power on, u-boot looks for that script in the located RFS at the flash and loads it into its environment. Then it starts the corresponding boot process.

To realize this behavior the **bootcmd** is defined to

- **run import.uboot_script && \$bootx**

The variable bootx presents the boot type and is defined to

- **flash_boot** as default

6.2.1 Implemented commands

Using the supported commands assumes either a previous saved environment based on the uboot_script or importing the uboot_script either by loading from flash or remote, like TFTP or NFS.

Importing from flash assumes a suitable RFS is located at the eMMC and looks like from the u-boot prompt as

- **run import.uboot_script**

The table lists some of the interested commands supported by the uboot_script.

Commando	loading mode	Purpose	active
update_uboot	NFS, TFTP	updating u-boot	y
update_spl	NFS, TFTP	updating spl	y
update_cm3	NFS, TFTP	updating cm3 firmware	y
start_cm3	local	loading and starting cm3 firmware from nor flash	n
delete_env	local	deleting environment in nor flash	y
net_boot	NFS	booting from NFS	y
restore_sys	NFS, TFTP	restore system	y
update_rfs	NFS, TFTP	updating RFS	y

Using a command needs setting some variables. These can be one or more of the following variables.

Variable	Value (def. → *)	depends on
ip-method	<ul style="list-style-type: none"> • static • dhcp* 	
lmode	<ul style="list-style-type: none"> • nfs • tftp 	
ipaddr netmask	ipaddr for target network mask	ip-method(static)
serverip	ipaddr of host	
nfsroot	location inside the exported <NFS_SHARE>	lmode(nfs)
bootdir	<ul style="list-style-type: none"> • /boot • location inside <TFTP_DIR> 	<ul style="list-style-type: none"> ➤ net_boot ➤ lmode tftp

6.3 Using U-Boot to change boot device or update parts of the system

This chapter describes how U-Boot has to be setup for updating and booting.

Performing of any of the tasks requires the IP-address of the host. For that, you have to identify the IP-address, first.

At a Debian host you get the address by entering **sudo ip addr** in the terminal.

Take the IP address of the corresponding network adapter and write down it for later use.

6.3.1 Boot setup and updating the RFS

Updating of the RFS and U-Boot is possible by using NFS as well by TFTP. For the RFS we recommend NFS, because it is much faster.

With the next sections we describe updating of the following images

- RFS → emtrion-image-sbc-rzn1d(-rt).tar.bz2
- U-Boot → u-boot.img
- SPL → spl-rel.spkg

To get ahead we assume the host system is prepared as described in 4.2 by the following requirements

- installed NFS, with **<NFS_SHARE>** is exported
- installed TFTP, with the tftp directory at **<TFTP_DIR>**
- emtrion's update mechanism emPURS, **<NFS_RESTORE>**

6.3.1.1 Updating of an old U-Boot with a version not included the string sbc-rzn1d

As mentioned in the section 4.2.10, while a flashed U-Boot at the SBC-RZN1D does not contain at least the string **sbc-rzn1d** in its version you have to update the SPL, U-Boot and the target's RFS. Below is a short instruction guide.

Updating of the SPL and U-Boot can be performed as before. First the SPL, immediately following the U-Boot.

- provide the binaries spl-rel.spkg and u-boot.img at the **<TFTP_DIR>**
- power on the SBC-RZN1D and stop booting
- enter the command sequence below at the U-Boot prompt
 - setenv serverip [ip-address of linux host]
 - setenv ip-method dhcp
 - run update_spl
 - run update_uboot
 - env default -a -f
 - saveenv
 - res

Updating the RFS we assume you have already installed emPURS as described in 4.2.9.

- provide the interested RFS at **<NFS_RESTORE>/images**
- enter the command sequence below at the U-Boot prompt
 - `setenv lmode nfs`
 - `dhcp`
 - `setenv serverip [ip-address of linux host]`
 - `setenv nfsroot <NFS_RESTORE>`
 - `${lmode} ${loadaddr} ${nfsroot}/uboot_script`
 - `env import -t ${loadaddr} ${filesize}`
 - `run update_rfs`

6.3.2 Updating the RFS (using NFS)

emtrion's update mechanism expects a specific image name. This name is defined to **emtrion-image-sbc-rzn1d.tar.bz2**. Subsequently, any interested RFS image has named to that one, while copying to the directory **images** in **<NFS_RESTORE>**.

```
cp *.tar.bz2 <NFS_RESTORE>/images/ image-sbc-rzn1d.tar.bz2.
```

Now you can use the following commands in the U-Boot prompt:

```
U-Boot # run import.uboot_script
U-Boot # setenv lmode nfs
U-Boot # setenv nfsroot <NFS_RESTORE>
U-Boot # setenv ip-method [dhcp or static]
U-Boot # setenv ipaddr [ip address for device, only needed for static ip]
U-Boot # setenv netmask [netmask for device, only needed for static ip]
U-Boot # dhcp [only needed by ip-method dhcp]
U-Boot # setenv serverip [ip-address of linux host]
```

After these settings you can start either the restore or update process. In comparison to the first one, the second one does not delete the partition of the eMMC.

Enter the command

- **run restore_sys**

or

- **run update_rfs**

This starts the update process. Please be patient as the process of fetching the root filesystem image via network and decompressing it to the flash storage can take a few minutes.

6.3.3 Updating of U-Boot bootloader (using TFTP)

Attention: If the board is turned off while updating the bootloader or another error occurs, the board will be rendered unusable to you. Please only update the bootloader if you are explicitly instructed to do so by emtrion.

If you have generated a new bootloader image **u-boot.img** as described in 5.2, copy it to the folder **<TFTP_DIR>** at the linux host. Otherwise you can use the image delivered by emtrion . Then you can start the update process with the following command in U-Boot prompt:

```
U-Boot # run import.uboot_script
U-Boot # setenv lmode tftp
U-Boot # setenv bootdir "./"
U-Boot # setenv ip-method [dhcp or static]
U-Boot # setenv ipaddr [ip-address for device, only needed for static ip]
U-Boot # setenv netmask [netmask for device, only needed for static ip]
U-Boot # dhcp [only needed by ip-method dhcp]
U-Boot # setenv serverip [ip-address of linux host]
U-Boot # run update_uboot
```

6.3.4 Updating of U-Boot SPL (using TFTP)

Attention: If the board is turned off while updating the bootloader or another error occurs, the board will be rendered unusable to you. Please only update the bootloader if you are explicitly instructed to do so by emtrion.

It might be necessary to update the U-Boot SPL. The SPL binary **spl-rel.spkg** has to be in the folder **<TFTP_DIR>** at the linux host. The current SPL binary is delivered by emtrion, too. Then you can start the update process with the following command in U-Boot prompt:

```
U-Boot # run import.uboot_script
U-Boot # setenv lmode tftp
U-Boot # setenv bootdir "./"
U-Boot # setenv ip-method [dhcp or static]
U-Boot # setenv ipaddr [ip-address for device, only needed for static ip]
U-Boot # setenv netmask [netmask for device, only needed for static ip]
U-Boot # dhcp [only needed by ip-method dhcp]
U-Boot # setenv serverip [ip-address of VM]
U-Boot # run update_spl
```

6.3.5 Booting

The default boot device in U-Boot is determined by the variable "bootx". If you want to set up one of the following boot options as a default you have to set "bootx" to the command mentioned below.

6.3.6 Boot from on-board flash

This is the default boot option configured when you receive the developer kit from emtrion and is defined as follow.

```
bootx=flash_boot
```

```
bootcmd=run import.uboot_script && $bootx
```

To start it manually simply use this command:

```
U-Boot # run run import.uboot_script && run flash_boot
```

6.3.7 Boot via network using a NFS share

We assume there is a decompressed RFS image available at **<NFS_ROOTFS>**.

In order to boot via network you have to perform the following commands in U-Boot:

```
U-Boot # run import.uboot_script
U-Boot # setenv nfsroot <NFS_ROOTFS>
U-Boot # setenv bootdir /boot
U-Boot # setenv ip-method [dhcp or static]
U-Boot # setenv ipaddr [ip-address for device, only needed for static ip]
U-Boot # setenv netmask [netmask for device, only needed for static ip]
U-Boot # saveenv
U-Boot # dhcp [only needed by ip-method dhcp]
U-Boot # setenv serverip [ip-address of linux host]
U-Boot # run net_boot
```

Now the board should *boot* via network using the NFS-share **<NFS_ROOTFS>** in the linux host.

If you want to make booting `net_boot` persistent for any reason, you can do that dependent on the `ip-method` as below:

- dhcp:
 - `setenv bootcmd 'run import.uboot_script && dhcp && run net_boot'`
- static:
 - `setenv bootcmd 'run import.uboot_script && run net_boot'`

now make it persistent

- `saveenv`

back to `flash_boot` do

- `setenv bootcmd 'run import.uboot_script && $bootx'`
- `saveenv`

7 SDK

In order to develop applications outside the Yocto build system (= outside the directory `<BUILD_DIR>`) you need to set up your host development system. For this purpose the YP offers several installation methods.

The bitbaking process described in chapter 5.2 creates an SDK installer containing the toolchain and the sysroot, which includes and matches the target RFS. The installer is stored in

`<BUILD_DIR>/emtrion/machines/sbc-rzn1d/tmp/deploy/sdk`

But you don't have to do the steps of bitbaking in chapter 5.2 to create the SDK installer yourself while emtrion has done that for you.

We assume you already have downloaded the SDK installers to `<DWL_DIR>`.

7.1 Installing the SDK

NOTE: emtrion has already done this step for its customers. You find the installed SDK in the directory `$SDK_DIR`.

In order to install the SDK, go to

`<BUILD_DIR>/emtrion/machines/sbc-rzn1d/tmp/deploy/sdk`

or

`<DWL_DIR>/SDKs/`

- ❖ none-RT
- ❖ RT

and execute the interested installer

`./poky-glibc-x86_64-emtrion-image-sbc-rzn1d(-systemd)-sdk-armv7vet2hf-vfpv4d16-sbc-rzn1d-toolchain-3.1.2.sh`

You are asked for the installation directory. You can either accept the suggested one or create one yourself.

7.1.1 Setting up the SDK environment

Before you can start developing apps you have to setup the environment. For that purpose a script is installed during the installation process of the SDK. The script is stored in the SDK's directory of `<SDK_DIR>`.

Performing the setup procedure, the script has to be sourced as follows.

source <SDK_DIR>/environment-setup- armv7vet2hf-vfpv4d16-poky-linux-gnueabi

The environment is only valid in the context of the terminal where this script has been called.

7.1.2 SDK Root Filesystem

Note, that in the directory **<SDK_DIR>/sysroots/armv7vet2hf-vfpv4d16-poky-linux-gnueabi** you will find the unpacked SDK Root Filesystem.

In your development phase it's recommended to use this SDK root filesystem on the target by setting the variable *nfsroot* accordingly based on the U-Boot environment set in 6.3.7

Boot the target and stop the boot process by hitting a key. Then set the *nfsroot* variable

setenv nfsroot <SDK_DIR>/sysroots/armv7vet2hf-vfpv4d16-poky-linux-gnueabi

Then if desired save the environment

saveenv

Now you can boot the target

dhcp [only needed by ip-method dhcp]

run net_boot

and start developing and debugging on the target.

8 Further Information

8.1 Online resources

Further information can be found on the emtrion support pages.

<https://support.emtrion.de>

8.2 We support you

emtrion offers different kinds of services, among them Support, Training and Engineering. Contact us at sales@emtrion.com if you need information or technical support.