www.vscom.de

OnRISC
User Manual
Edition: September 2016

Vision Systems GmbH
Tel: +49 40 528 401 0
Fax: +49 40 528 401 99
Web: www.visionsystems.de
Support: faq.visionsystems.de

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

Copyright Notice

Trademarks

VScom is a registered trademark of Vision Systems GmbH. All other trademarks and brands are property of their rightful owners.

Disclaimer

Vision Systems reserves the right to make changes and improvements to its product without providing notice.

Vision Systems provides this document “as is”, without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Vision Systems reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Vision Systems assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Contents

List of Figures

List of Tables

1. Introduction

1.1. OnRISC Family

The OnRISC is an ARM-based RISC industrial embedded computer family. The great variety of interfaces like LAN, CFast, microSD/SD, USB, CAN, I²C, serial interface and digital I/O makes it easy to connect various industrial devices to the OnRISC. Some new OnRISC devices provide graphical display (VS-860 has built-in 8" LCD display).

Compact dimensions and DIN Rail mounting capability make the OnRISC to a space saving and flexible mounting industrial computer. It is feasible to be installed even in space limited environments.

Due to RISC based architecture the OnRISC has very small power consumption, so fanless heat dissipation is possible. Working in an extended temperature range the OnRISC can be used under harsh industrial conditions. Therefore the OnRISC is downright designed for industrial automation. Refer to the Hardware Manual for exact characteristics.

The embedded computer runs full-featured Debian GNU/Linux on ARM operating system. With Debian's repository database it is easy to install and update the free software on the OnRISC. The OnRISC is capable to act directly as software development host, Web, Mail, Print and Database server or as desktop computer with X11 window manager and many more.

1.2. How to Read the Manual?

Hardware Manual should be consulted first to understand, how the device is going to be powered. Section "Getting Started" will get you from power on till log on and basic network configuration. Section "Debian Maintenance Notes" describes how to search/install/remove Debian packages. Section "Network Services and Tools Provided by OnRISC" shows how to setup various network services like SSH, ser2net, socketcand etc.

Software development related sections are spread over the manual and touching different aspects of embedded software development for OnRISC devices. Section "Software Development" gives step-by-step instructions, how to setup development environment, get source code for the Linux kernel etc. Section "Hardware API" gives overview of hardware interfaces and how they can be accessed in software. Section "Recommended Books" provides a list of books about Linux administration and programming.

Section "Buildroot" introduces another embedded Linux distribution built from scratch, that can be used instead of Debian.

OnRISC Model	Baltos iR 5221 Baltos iR 3220	Baltos iR 2110	Baltos 1080
CPU	AM335x (ARM Cortex-A8)	AM335x (ARM Cortex-A8)	AM335x (ARM Cortex-A8)
RAM	256MB DDR3	256MB DDR3	256MB DDR3
Flash Memory on Board	256MB NAND	256MB NAND	256MB NAND
Serial Interfaces	2 x RS232/RS422/RS485 (16C750)	1 x RS232/RS422/RS485 (16C750)	8 x RS232 (FTDI)
CAN Interface on board	up to 1 x	N.A.	N.A.
Digital I/O channels	4 x inputs, 4 x outputs	N.A.	8 x inputs, 8 x outputs
CFast slot	N.A.	N.A.	N.A.
SD-Slot	1 x external slot	1 x external microSD-slot	1 x internal microSD-slot
USB	2 x USB 2.0 as Host, up to 1 x as OTG	1 x USB 2.0 as Host	N.A.
Expansion Slot	Mini PCI Express	N.A.	N.A.
Ethernet	1 x Gbit, up to 4 x 100Mbit switch	1 x Gbit, 1 x 100Mbit	1 x Gbit (WAN)
I²C bus	1 x	N.A.	N.A.
RTC	1 x	1 x	1 x
Watchdog Timer	1 x	1 x	1 x
Display	N.A.	N.A.	N.A.

Table 1: OnRISC Products Based on OMAP3 am335x SoC

OnRISC Model	OnRISC 113 OnRISC 213	OnRISC 413 OnRISC 813	OnRISC CAN
CPU	AM335x (ARM Cortex-A8)	AM335x (ARM Cortex-A8)	AM335x (ARM Cortex-A8)
RAM	256MB DDR3	256MB DDR3	256MB DDR3
Flash Memory on Board	256MB NAND	256MB NAND	256MB NAND
Serial Interfaces	up to 2 x RS232/RS422/RS485 (16C750)	up to 8 x RS232/RS422/RS485 (FTDI)	N.A.
CAN Interface on board	N.A.	N.A.	1 x
Digital I/O channels	N.A.	N.A.	N.A.
CFast slot	N.A.	N.A.	N.A.
SD-Slot	1 x internal microSD-slot	1 x internal microSD-slot	1 x internal microSD-slot
USB	1 x USB 2.0 as Host	1 x USB 2.0 as Host	1 x USB 2.0 as Host
Expansion Slot	N.A.	N.A.	N.A.
Ethernet	1 x Gbit (WAN)	1 x Gbit (WAN)	1 x Gbit (WAN)
I²C bus	N.A.	N.A.	N.A.
RTC	1 x	1 x	1 x
Watchdog Timer	1 x	1 x	1 x
Display	N.A.	N.A.	N.A.

Table 2: OnRISC Products Based on OMAP3 am335x SoC

OnRISC Model	VS-860
CPU	AM3517 (ARM Cortex-A8)
RAM	256MB DDR2
Flash Memory on Board	256MB NAND
Serial Interfaces	2 x RS232/RS422/RS485 (FTDI)
CAN Interface on board	1 x
Digital I/O channels	N.A.
CFast slot	1 x
SD-Slot	1 x external slot
USB	2 x USB 2.0 as Host, 1 x as OTG
Expansion Slot	Mini PCI Express
Ethernet	2 x
I²C bus	N.A.
RTC	1 x
Watchdog Timer	1 x
Display	built in LCD

Table 3: OnRISC Products Based on OMAP3 am3517 SoC

2. Getting Started

2.1. Connect to OnRISC via Serial Link

Connect the OnRISC to the serial port of your PC and start a terminal software (HyperTerminal, ZOC[1], minicom etc) with 115200,8,n,1 settings (no hardware/software handshake is needed. Set the terminal type according to Section 2.2). Insert a SD/microSD-card with a preinstalled system (refer to Section 4). Power your OnRISC according to the Hardware Manual. You'll see Linux booting. After the boot procedure you'll be asked to log in. As only super user (root) is available use following credentials to login:

```
Debian login:  root
Password:  linux
```

2.2. Terminal Type

Terminal type is defined in the environment variable `TERM` and is set to `TERM=linux` by default. The terminal type of your terminal application (HyperTerminal, ZOC, minicom etc.) should be set to the same type to interact correctly with the OnRISC console. If `linux` terminal type is not available in your software, `vt100` can be used instead. To do this add following line to the `~/.bashrc`:

```
export TERM=vt100
```

[1] www.emtec.com

2.3. Configure Network

Now you can configure network interfaces by editing `/etc/network/interfaces`. The IP addresses for `eth0`, `eth1` and `wlan0`[2] are statically assigned by default (see the Listing below). Please refer to Section 5.1 for detailed hardware network interface mapping, i.e. what network device name corresponds to what physical connector.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.254.254
        netmask 255.255.255.0

# The secondary network interface
auto eth1
iface eth1 inet static
        address 192.168.253.254
        netmask 255.255.255.0

# The wireless interface
#auto wlan0
iface wlan0 inet static
        address 192.168.127.254
        netmask 255.255.255.0
        wpa-driver nl80211
        wpa-conf /etc/wpa_supplicant.conf
```

Listing 1: /etc/network/interfaces

The wireless interface `wlan0` will be configured with the `wpa_supplicant` utility (see Section 5.2).

Setup gateway and DNS server Assuming your router has IP address 192.168.254.1 the OnRISC will be configured in the following way:

1. change `eth1` section of `/etc/network/interfaces` file
   ```
   auto eth1
   iface eth1 inet static
   address 192.168.254.254
   netmask 255.255.255.0
   gateway 192.168.254.1
   ```
2. insert following line to the `/etc/resolv.conf`[3]
   ```
   nameserver 192.168.254.1
   ```
3. execute `/etc/init.d/networking restart`

[2] to activate wlan0 uncomment the `#auto wlan0` line in `/etc/networking/interfaces`
[3] see `man resolv.conf` for explanations

3. Supported Linux Distributions

Following Linux distributions are provided for OnRISC devices:

- Debian
- Buildroot
- Yocto

Each distribution has its own advantages and disadvantages, so one has to consider, what properties are important for the project in question[4]. Below you'll find comparison of these distributions and suggestions for bringing your project on OnRISC.

3.1. Debian

Debian is well known and established Linux distribution in the desktop/server world. So working with its ARM port doesn't really differ from working with your PC. Debian for ARM provides same packages and tools. So it is best suited to start developing your software with it. Open Source dependency installation from Debian repositories is very fast and can be done at any time. And if you design your software to create a *.deb package, you'll get an update mechanism right away. But this flexibility has its price: your root file system exceeds the size of 1GB very quickly depending on required software dependencies (libraries, frameworks). This makes it almost unsuitable for NAND flash based projects.

3.2. Buildroot

Buildroot is a set of Makefiles and patches, that compiles root file system from scratch. So initial stage can take more than hour to compile depending on hardware setup of your PC. Buildroot doesn't provide binary package management system like Debian, so adding new packages will require rewriting root file system on target medium (NAND or SD/microSD-card) again. But particularly this approach leads to relative small core system. Depending on what software dependencies your software has, you can get an initramfs image (kernel and rootfs) of about 10MB. Such approach makes firmware update procedure quite straight forward: start the system and just replace `kernel-fit.itb`. That's all.

3.3. Yocto

Yocto is also a set of Makefiles and patches, that compiles root file system from scratch. But compared to Buildroot it also creates re-distributable binary packages (rpm ,deb), so that one can easily update a rootfs in production.

[4] These slides provided by Mind compare Debian and Buildroot as embedded Linux distributions: http://www.mind.be/content/Presentation_Emdedded-distro-shootout.pdf

3.4. Distribution Choice

First of all a decision must be made, what medium to use NAND flash or SD/microSD-card. If NAND flash, then Buildroot or Yocto are the right choices. For SD/microSD-card you still have alternative and other factors play role. But regardless of distribution choice, it is easier to start developing with Debian. Because you can experiment with different libraries/frameworks and so determine required set of software dependencies. This set can then be used either to configure Buildroot, Yocto or vsdebootstrap (script, that creates custom Debian root file system on your host PC).

4. Software Configuration

The OnRISC comes with a preinstalled Debian GNU/Linux on ARM[5] operating system. The system image (see Section 4.5), that can be either downloaded or obtained as part of the Starter kit, provides necessary tools and services to start with application development, various services such as ssh, RFC2217 etc.

This image can be downloaded from our FTP server[6] and can be copied to the SD/microSD-card as described in Appendix D.1.

Alternatively Buildroot (Section 9) can be used to create the root file system.

4.1. Init System

Debian 8 uses `systemd`[7] as default init system. `systemd` differs in many points from System V like init system, but it maintains compatibility layer, so that legacy start-up scripts still can be used. For example you can still configure network interfaces via `/etc/network/interfaces` (default for our Debian 8 image), but the new approach were to remove `/etc/network/interfaces` and configure network via `networkd` (see Arch-Linux wiki[8] for examples)

4.2. Boot Device Sequence

Baltos systems will first check NAND device, if it contains a valid boot image, i.e. the first NAND partition is not empty. If the first partition is not empty, the system starts bootloader from NAND, if it is empty, MMC device will be searched for MLO file. Regardless of what device has started the bootloader, the final decision where to start the kernel from, will be taken by the bootloader's script.

[5] http://www.debian.org/ports/arm/
[6] ftp://ftp.visionsystems.de/pub/multiio/OnRISC/
[7] https://en.wikipedia.org/wiki/Systemd
[8] https://wiki.archlinux.org/index.php/Systemd-networkd

4.2.1. Booting from SD/microSD-card

The system images for OMAP3 based devices have two partitions:

1. FAT[9] partition having files need to initialize and boot the system (`MLO`, `u-boot.img`, `uEnv.txt` and `uImage` or `kernel-fit.itb`)
2. ext4 partition having Debian root file system

The OMAP CPU automatically loads SPL[10] (`x-loader/MLO`) from the FAT partition and then the code in SPL takes over and loads U-Boot[11] (`u-boot.img`), that takes care of Linux kernel (`uImage` or `kernel-fit.itb`). Please refer to this wiki for detailed information about OMAP boot process Bootloader Project.

4.2.2. Booting from NAND

Baltos has following NAND partitions:

```
#define MTDPARTS_DEFAULT    "mtdparts=omap2-nand.0:128k(SPL)," \
                            "128k(SPL.backup1)," \
                            "128k(SPL.backup2)," \
                            "128k(SPL.backup3)," \
                            "1920k(u-boot)," \
                            "-(UBI)"
```

Listing 2: NAND Partition Table

Partitions SPL - SPL.backup3 have the same MLO as on the SD/microSD-card, u-boot holds U-Boot, UBI MTD partition holds further UBIFS[12] formatted partitions like kernel and rootfs (see Section 9.5.2 for details).

4.3. Swapping and Logging

Due to the fact that the flash memory has a finite number of erase-write cycles it is very important to reduce them. Many applications use logging for information, recovery and debugging purposes, this can lead to frequent flash usage. There are several possibilities to avoid this:

1. use external HDD attached via USB and redirect swapping and logging to it
2. disable swapping[13] (remove swap entry in the `/etc/fstab`) and logging where it is possible
3. redirect the log stream via network[14]

To receive the log messages under Linux you can use your existing `syslog` utility, for Windows you can use Kiwi Syslog Daemon[15] or any other Syslog daemon.

[9] this partition must have bootable flag set
[10] Secondary Program Loader
[11] http://www.denx.de/wiki/U-Boot
[12] http://en.wikipedia.org/wiki/UBIFS
[13] To list swapping devices execute `cat /proc/swaps`
[14] The Debian Administrator's Handbook
[15] www.kiwisyslog.com

```
#define NANDARGS \
        "mtdids=" MTDIDS_DEFAULT "\0" \
        "mtdparts=" MTDPARTS_DEFAULT "\0" \
        "nandargs=setenv␣bootargs␣console=${console}␣" \
                "${optargs}␣" \
                "${mtdparts}␣" \
                "root=${nandroot}␣" \
                "rootfstype=${nandrootfstype}\0" \
        "nandroot=ubi0:rootfs␣rw␣ubi.mtd=5\0" \
        "nandrootfstype=ubifs␣rootwait=1\0" \
        "nandboot=echo␣Booting␣from␣nand␣...;␣" \
                "run␣nandargs;␣" \
                "setenv␣loadaddr␣0x81000000;␣" \
                "ubi␣part␣UBI;␣" \
                "ubifsmount␣ubi0:kernel;␣" \
                "ubifsload␣$loadaddr␣kernel-fit.itb;" \
                "ubifsumount;␣" \
                "bootm␣$loadaddr\0"
```

Listing 3: U-Boot: NAND Settings

4.4. Activating and Deactivating Services

Some services will be started as daemons at system startup and hence reduce the amount of free memory and increase the boot time. The ways to activate/deactivate services at boot time are extensively covered in Sections System Boot and The inetd Super-Server of "The Debian Administrator's Handbook".

4.5. System Image

The complete system image contains lots of programs and libraries. It contains a development environment consisting of the gcc toolchain and vim-tiny text editor. This image was created with vsdebootstrap (see Section 8 for details).

4.5.1. Program Overview

The complete image provides among others the following utilities:

- Software Development
  - gcc
  - CMake
  - git
- Network
  - ssh (server and client)

  - netcat
  - ser2net RFC2217 server
  - socketcand
- Desktop:
  - XFCE desktop

5. Network Services and Tools Provided by OnRISC

The OnRISC can be accessed via Ethernet for remote usage and file sharing. For this purpose there are several services such as `telnet` or `ssh` installed and preconfigured. For WLAN configuration `wpa_supplicant` and `iw` are included in the distribution.

5.1. LAN Configuration

5.1.1. Baltos

Baltos systems have following network interfaces:

1. WAN (`eth1`) - all devices
2. LAN (`eth0`) - iR devices and VS-860

WAN is connected to a single port PHY (Atheros 8035). Linux kernel automatically shows cable insertion/removal status on the console.

```
cpsw 4a100000.ethernet eth1:  Link is Down
```

or

```
cpsw 4a100000.ethernet eth1:  Link is Up - 1Gbps/Full - flow control off
```

LAN configuration differs across devices. In iR 5221/3220 devices LAN (`eth0`) is connected to a switch chip (ICPlus IP175D). As shown in Figure 1 on page 20 `eth0` is connected to one of the switch ports, hence has always cable inserted status. The LAN of iR 2110 is connected to a single port PHY (ICPlus IP101A/G) and behaves like WAN interface. The only difference is the maximum speed of 100Mbit/s. And finally Baltos 1080, OnRISC 113/213/413/813/CAN have WAN interface only i.e. `eth1`.

VLAN The network controller in Baltos systems is implemented as a switch device (CPSW). In order to use both slaves as a separate network interfaces special feature called Dual Emac is used. Dual Emac uses VLAN ID 1 and 2 in order to separate `eth0` and `eth1`, hence you should not add these VLAN IDs via `vconfig` or `iproute2`.

If you are forced to use VLAN ID 1 and/or 2, you'll have to change default VLAN IDs in the DTS file corresponding to your device, recompile and install the kernel. In the Listing below you can see the field `dual_emac_res_vlan` specifying default VLAN IDs. Just change 1 and 2 to for example 4093 and 4094.

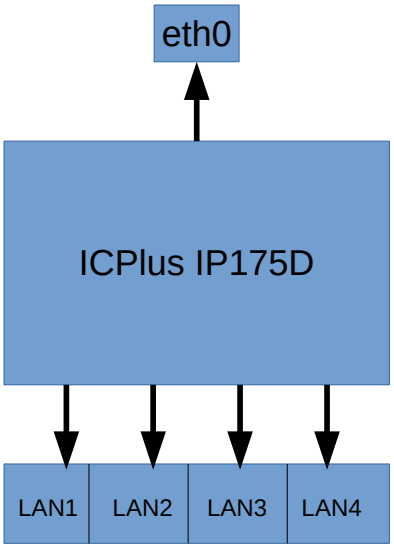

Figure 1: iR 5221 LAN Configuration

```
&cpsw_emac0 {
        phy_id = <&davinci_mdio>, <0>;
        phy-mode = "rmii";
        dual_emac_res_vlan = <1>;
};

&cpsw_emac1 {
        phy_id = <&davinci_mdio>, <7>;
        phy-mode = "rgmii-txid";
        dual_emac_res_vlan = <2>;
};
```

Listing 4: DTS: Dual Emac Configuration

5.2. WLAN Configuration

Wpa_supplicant[16] is a WPA Supplicant for Linux, BSD, Mac OS X, and Windows with support for WEP, WPA and WPA2 (IEEE 802.11i / RSN). Supplicant is the IEEE 802.1X/WPA component that is used in the client stations. It implements key negotiation with a WPA Authenticator and it controls the roaming and IEEE 802.11 authentication/association of the WLAN driver.

5.2.1. Managed Wireless Network (Wpa_supplicant)

Wpa_supplicant uses `/etc/wpa_supplicant.conf` file for its configuration (see the Listing below).

```
ap_scan=1

# no encryption
network={
   ssid="TEST"
   key_mgmt=NONE
}
# WEP encryption
network={
   ssid="TESTWEP"
   key_mgmt=NONE
   wep_key0=xxxxxxxxxxxxxxxxxxxxx
   wep_tx_keyidx=0
   auth_alg=SHARED
}
# WPA/WPA2 encryption
network={
   ssid="TESTWPA2"
   proto=WPA RSN
   key_mgmt=WPA-PSK
   pairwise=CCMP TKIP
   group=CCMP TKIP
   psk="xxxxxxxxxxxxxx"
}
```

Listing 5: /etc/wpa_supplicant.conf

WLAN interface is automatically configured on system's startup (see Section 2.3). To test the configuration just run:

```
/etc/init.d/networking restart
```

Wpa_supplicant can also be called manually to investigate configuration problems:

```
wpa_supplicant -iwlan0 -c/etc/wpa_supplicant.conf -dd -Dnl80211
```

Further information about configuring the WLAN interface can be taken from wpa_supplicant's manual page (`man wpa_supplicant`) .

[16] http://hostap.epitest.fi/wpa_supplicant/

5.2.2. Ad-hoc Wireless Network

Ad-hoc connections will be managed via iw tool. Following example shows, how to connect to an unencrypted Ad-hoc host “foo”:

```
iw wlan0 connect foo
```

To connect to a WEP encrypted host invoke:

```
iw wlan0 connect foo keys 0:abcde d:1:0011223344
```

5.2.3. Supported WLAN Hardware

Model	Interface	Kernel Symbols	Firmware (Buildroot Symbols)
Jorjin WG7833	SDIO	WL18XX, WLCORE_SDIO	BR2_PACKAGE_LINUX_FIRMWARE_TI_WL18XX
TP-LINK TL-WN272N V3	USB	RT2800USB_RT53XX	BR2_PACKAGE_LINUX_FIRMWARE_RALINK_RT2XX
SparkLAN WPER-172GN	USB	RT2800USB_RT53XX	BR2_PACKAGE_LINUX_FIRMWARE_RALINK_RT2XX

Table 4: Supported WLAN Hardware

5.3. GSM Support

Using USB connector or PCI Express Mini Card adapter described in the Hardware Manual you can attach a GSM card. The system was tested with following GSM/UMTS/HSPA cards:

Vendor	Model	3G	4G	GPS type	Connector	VID/PID
Huawei	MU609	yes	no	passive antenna	mPCIe	12d1:1573
Huawei	ME909u-521	yes	yes	passive antenna	mPCIe	12d1:1573
Sierra Wireless	AirPrime MC7304	yes	yes	active antenna	mPCIe	1199:68c0
Quectel	UC20	yes	no	passive antenna	mPCIe	05c6:9003
SIMcom	SIM5360E	yes	no	none	mPCIe	05c6:9000
Huawei	E353	yes	no	none	USB	12d1:1506

Table 5: Supported Modems

Please read Section 7.3. It describes, how mPCIe slot can be turned on and off.

5.3.1. PPP

Special udev rules in `/etc/udev/rules.d/98-vscom.rules` create `/dev/atcmd0` symlink to card's command port, that is used in the scripts below. This allows the user to use any supported card without changing its device name in the scripts. Following configuration files were prepared for modem usage:

- `/etc/chatscripts/gprs` (your provider's Access Point Name (APN) is stored here)
- `/etc/ppp/peers/gprs`

To activate a GSM connection configure proper APN and invoke

```
pppd file /etc/ppp/peers/gprs
```

You can find the supported AT command reference[17] on the CD .

5.3.2. ModemManager

Starting from version 8 Debian provides ModemManager[18] with QMI[19]/MBIM[20] support. These protocols/drivers provide higher data throughput than PPP. Below you'll find some usage examples:

List detected modems:

```
# mmcli -L

Found 1 modems:  /org/freedesktop/ModemManager1/Modem/0 [Huawei Technologies Co., Ltd.]  MU609
```

Show modem's status:

```
# mmcli -m 0
/org/freedesktop/ModemManager1/Modem/0 (device id '4a1d302560babc0bc24d6a46e1d8c56740045b8f')
```

[17] documentation/2130617_Supported_AT_Command_Reference-v2.4.pdf
[18] http://www.freedesktop.org/wiki/Software/ModemManager/
[19] http://www.freedesktop.org/wiki/Software/libqmi/
[20] http://www.freedesktop.org/wiki/Software/libmbim/

```
-------------------------
Hardware | manufacturer:  'Huawei Technologies Co., Ltd.'
| model:  'MU609'
| revision:  '12.105.29.00.00'
| supported:  'gsm-umts'
| current:  'gsm-umts'
| equipment id:  '323432046447031'
-------------------------
System | device:  '/sys/devices/ocp/47400000.usb/47401400.usb/musb-hdrc.1.auto/usb1/1-1/1-1.3'
| drivers:  'option1, cdc_ether'
| plugin:  'Huawei'
| primary port:  'ttyUSB2'
| ports:  'ttyUSB0 (at), eth2 (net), ttyUSB2 (at), ttyUSB3 (gps), ttyUSB4 (at), ttyUSB1 (qcdm)'
-------------------------
Numbers | own :  '+xxxxxx'
-------------------------
Status | lock:  'none'
| unlock retries:  'sim-pin (3), sim-pin2 (2), sim-puk (10), sim-puk2 (10)'
| state:  'disabled'
| power state:  'on'
| access tech:  'unknown'
| signal quality:  '0' (cached)
-------------------------
Modes | supported:  'allowed:  3g; preferred:  none
| allowed:  2g; preferred:  none
| allowed:  2g, 3g; preferred:  none'
| current:  'allowed:  2g, 3g; preferred:  none'
-------------------------
Bands | supported:  'unknown'
| current:  'unknown'

-------------------------
IP | supported:  'ipv4'
-------------------------
3GPP | imei:  '11122222111'
| enabled locks:  'none'
| operator id:  'unknown'
| operator name:  'unknown'
| subscription:  'unknown'
| registration:  'unknown'
-------------------------
SIM | path:  '/org/freedesktop/ModemManager1/SIM/0'
-------------------------
Bearers | paths:  'none'
```

Enable modem:

```
# mmcli -m 0 -e
successfully enabled the modem
```

Create simple connection:

```
# mmcli -m 0 --simple-connect="apn=internet.eplus.de"
successfully connected the modem
```

After the successful connection you can obtain IP address using `dhclient` and network interface mentioned in `mmcli -m 0` output (in the case of Huawei MU609 it is `eth2`):

```
# dhclient eth2
```

Disconnect:

```
# mmcli -m 0 --simple-disconnect
successfully disconnected all bearers in the modem
```

5.3.3. Connect Modem On Startup

There are many ways to activate the modem on startup. Below a method using `incron`[21] will be shown. First of all you'll have to install `incron`:

```
apt install incron
```

Open `/etc/incron.allow` and enter root, save and exit. Now root is allowed to perform incron jobs. To create modem job invoke:

```
incrontab -e
```

This command will open an editor, where you will enter following line, save and exit:

```
/dev/ IN_CREATE /usr/local/bin/startmodem.sh
```

After configuring `incron` create `/usr/local/bin/startmodem.sh` with following code:

```
#!/bin/sh

if [ -e /dev/atcmd0 ]; then
        sleep 60
        # enter PIN, if required
        #echo -e "at+cpin="1234"\r" > /dev/atcmd0
        /usr/sbin/pppd file /etc/ppp/peers/gprs
fi
```

Make file executable `chmod +x /usr/local/bin/startmodem.sh`

Now every time the system boots and mPCIe slot is activated `/usr/local/bin/startmodem.sh` will be started. `/usr/local/bin/startmodem.sh` waits for a minute to give ModemManager time to find the modem and then starts `pppd`. You can replace `pppd` invocation with `mmcli` as described in Section 5.3.2.

5.4. GPS

You can use various devices to get GPS data. So far following devices were tested:

1. VersaLogic VL-MPEu-G2E - dedicated mPCIe card with active GPS antenna support
2. GlobalSat BU-353S4 - USB GPS mouse
3. Sierra Wireless AirPrime MC7304 - mPCIe LTE modem with active GPS antenna support

5.4.1. GPS Device Setup

VersaLogic VL-MPEu-G2E VL-MPEu-G2E exports /dev/ttyACM0 device. Baudrate 9600b/s:

```
stty -F /dev/ttyUSB0 ispeed 9600 ospeed 9600
```

[21] http://inotify.aiken.cz/?section=incron&page=about&lang=en

GlobalSat BU-353S4 BU-353S4 exports /dev/ttyUSB0 device provided by pl2303 driver. Baudrate 4800b/s:

```
stty -F /dev/ttyUSB0 ispeed 4800 ospeed 4800
```

Sierra Wireless AirPrime MC7304 MC7304 provides GPS data on the `/dev/ttyUSB1` device. You'll have to active the GPS port prior to reading data:

```
echo 'AT!GPSTRACK=1,255,100,1000,5' > /dev/ttyUSB2
stty raw -F /dev/ttyUSB1; echo \$GPS_START >/dev/ttyUSB1
```

5.4.2. Reading Raw GPS Data

After configuring the required device you can just invoke `cat device` in order to get raw data:

```
cat /dev/ttyUSB0
```

You'll get similar output:

```
$GPGSA,A,3,12,15,25,32,10,,,,,3.4,2.3,2.5*33
$GPRMC,160059.000,A,5340.0274,N,00959.2111,E,0.48,38.52,090316,,A*5D
$GPGGA,160100.000,5340.0274,N,00959.2116,E,1,05,2.3,10.4,M,45.9,M,,0000*61
$GPGSA,A,3,12,15,25,32,10,,,,,3.4,2.3,2.5*33
$GPRMC,160100.000,A,5340.0274,N,00959.2116,E,0.47,38.52,090316,,A*58
$GPGGA,160101.000,5340.0273,N,00959.2114,E,1,04,4.1,10.5,M,45.9,M,,0000*61
```

The meaning of these messages can be read here.

5.4.3. gpsd

gpsd project[22] provides various tools to manage GPS receivers and to provide their data over network.

gpsmon is a real-time GPS packet monitor and control utility. Execute following command to see GPS coordinates, satellite information etc.

```
gpsmon /dev/ttyUSB0
```

gpsd is interface daemon for GPS receivers. To start a daemon in foreground and listening on TCP port 5050 execute:

```
gpsd -b -G -N -S 5050 /dev/ttyUSB0
```

[22] http://www.catb.org/gpsd/

5.5. SSH

To access the OnRISC via SSH from Linux execute:

```
ssh root@192.168.254.254
```

To access OnRISC via SSH from Windows you need a ssh-client such as PuTTY[23]. To exchange files several tools could be used:

- scp (Linux) - secure copy tool
- pscp (Windows) - secure copy tool included in PuTTY distribution
- WinSCP[24] (Windows) - secure copy tool with graphical interface

For further information see:

```
man sshd
```

5.6. RFC2217

Internal serial interfaces of the OnRISC can be made accessible over network via RFC2217 protocol. `ser2net`[25] is a daemon, that provides such functionality. On the client side (PC) you need either a virtual COM port driver or an application/library communicating RFC2217 directly. For MS Windows OS you can use the VScom driver of NetCom Mini[26].

The installation requires following components to be installed:

- RFC2217 driver and NetCom UPnP Manager (is bundled with the driver)
- `vsupnpd` (provided with our system images, enabled by default)
- `ser2net` (provided with our system images, disabled by default)

`ser2net` will be configured via special configuration file (default `/etc/ser2net.conf`). This file contains following information:

```
SIGNATURE:sign_vs:VScom NetCom:111S:(C) VS Vision Systems GmbH 2010:1
5000:telnet:0:/dev/ttyO1:115200 sign_vs
5001:telnet:0:/dev/ttyO2:115200 sign_vs
```

SIGNATURE is a string that will be sent on RFC2217 driver's request to identify the device. Don't change this setting. After that you'll find the per serial interface configuration strings in following format:

```
<TCP port>:<state>:<timeout>:<device>:<options>
```

In the example the `/dev/ttyO1` device will be used in RFC2217 mode (telnet), listens on TCP port 5000 and the timeout function is disabled.

`vsupnpd` daemon parses `/etc/ser2net.conf` file and extracts TCP port numbers. This information will be announced via SSDP broadcasts, so that NetCom UPnP Manager can find and install the

[23] http://www.chiark.greenend.org.uk/~sgtatham/putty/
[24] http://winscp.net/eng/index.php
[25] https://sourceforge.net/projects/ser2net/
[26] http://www.vscom.de/download/multiio/Windows7/driver/NCMini-Install-7z.exe

serial interfaces. If OnRISC is behind a router and broadcasts are not allowed, you can manually add the device in the NetCom UPnP Manager. See NetCom Mini Manual for more details.

Please note that `ser2net` service is disabled by default. You need to activate using `rcconf` (see Section 4.4).

5.7. Socketcand

`socketcand` is a daemon that provides access to CAN interfaces on a machine via a network interface. The communication protocol uses a TCP/IP connection and a specific protocol to transfer CAN frames and control commands. The protocol specification can be found in `./doc/protocol.md`.[27]

The project provides Java API to develop your own client applications for `socketcand`. There is also a CAN sniffer software Kayak[28], that uses socketcand protocol, so that you can use OnRISC as a network based CAN sniffer at once.

Please note that `socketcand` service is disabled by default. You need to activate using `rcconf` (see Section 4.4).

5.8. GPIO over Modbus/TCP

Baltos GPIO can be made accessible over network via `modbusgpio` daemon. `modbusgpio` can be invoked with following parameter:

1. TCP port number - port number `modbusgpio` is listening on (required parameter)
2. IP address to bind daemon to (optional parameter)
3. debug enable option (optional parameter)

Example:

```
modbusgpio 502 debug
```

This command would start `modbusgpio`, that listens on port 502 and also prints debug messages on `stderr`.

The daemon implementation supports Discrete Inputs, Coils, Input Registers and Holding Registers. These are the most useful function codes (see Table 6 on page 29).

	Function	Decimal Code
Single bit access	Read Discrete Inputs	02
	Read Coils	01
	Write Single Coil	05
	Write Multiple Coils	15
16 bit access	Read Input Register	04

Table 6: Function Codes Modbus/TCP for Digital-I/O

Mapping of addresses to Digital-I/O signals are shown in following tables:

[27] https://github.com/dschanoeh/socketcand
[28] http://kayak.2codeornot2code.org/

Coil Address	Register Address	Signals	Interpretation
0 .. 7	0	Input 1 .. 8	External status 0: Input open 1: Input closed
8 .. 15		Output 1 .. 8	External status 0: Connect <n> to Ö 1: Connect <n> to S
32 .. 47	2	Data Direction	0: Input signal 1: Output signal
64 .. 79	4	Direction Option	0: Changeable 1: Fixed

Table 7: Mapping Modbus addresses to Input-/Output-signals for Baltos 1080

Coil Address	Register Address	Signals	Interpretation
0 .. 3	0	IN0 .. IN3	External status 0: low 1: high
4 .. 7		OUT0 .. OUT3	External status 0: low 1: high
32 .. 47	2	Data Direction	0: Input signal 1: Output signal
64 .. 79	4	Direction Option	0: Changeable 1: Fixed

Table 8: Mapping Modbus addresses to Input-/Output-signals for Baltos 5221/3220

For initial testing you can use QModBus[29] software to set/get GPIO values. Figure 2 shows how to set OUT2 to high and Figure 3 shows the result provided OUT2 is connected to IN2.

[29] http://qmodbus.sourceforge.net/

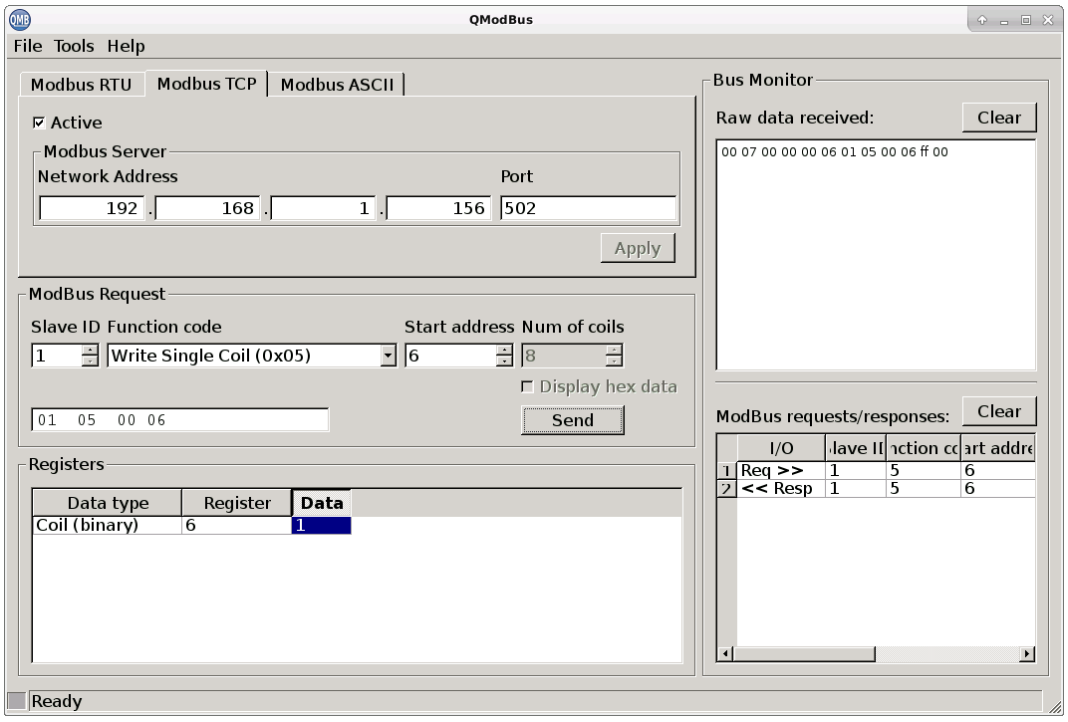

Figure 2: QModBus: Baltos iR 5221: Set OUT2 to High

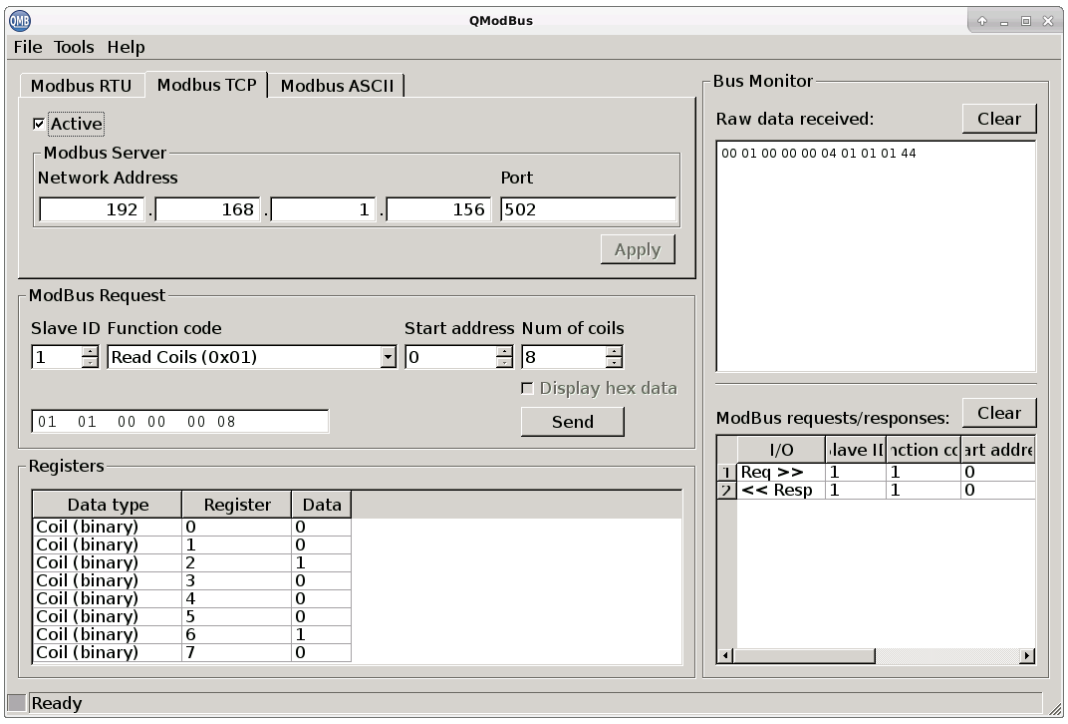

Figure 3: QModBus: Baltos iR 5221: Get all GPIOs

6. Software Development

6.1. Environment

6.1.1. Compile your software directly on the OnRISC

You can start programming directly on the OnRISC. The toolchain is already installed and GCC compiler will be invoked in the same way it is done on a desktop Linux. To modify files you can use `vi` or some other editor. This method is preferred, if your software has many dependencies.

6.1.2. Cross-compile your software on the PC

Debian 8.0 will be supplied with OMAP3 based devices. It is based on an arm-linux-gnueabihf 4.9.x toolchain. To install this toolchain on your development PC, you'll need to add emdebian repository. Add `deb http://emdebian.org/tools/debian/ jessie main` to your `/etc/apt/sources.list`. Then execute as root:

```
dpkg --add-architecture armhf
curl http://emdebian.org/tools/debian/emdebian-toolchain-archive.key | apt-key add -
apt-get update
apt-get install crossbuild-essential-armhf
apt-get install build-essential git debootstrap u-boot-tools
```

If your software has dependencies on other libraries provided by Debian, it is recommended to use Debian 8 on your development host. You can get cross-compiled libraries via `apt-get install package:armhf`. For example to install `libsocketcan` you need to invoke following command:

```
apt-get install libsocketcan-dev:armhf
```

After installation you'll see following packages appearing in your package data base:

```
# dpkg -l | grep libsocketcan

ii libsocketcan-dev 0.0.9+git20140207-1 armhf library to control some basic functions in SocketCAN from
userspace

ii libsocketcan2 0.0.9+git20140207-1 armhf library to control some basic functions in SocketCAN from
userspace
```

Following code sample uses `libsocketcan` to shutdown `can0` interface:

```
#include <stdio.h>
#include <libsocketcan.h>

int main(int argc, char **argv)
{
        printf("Hello␣world!\n");

        if (can_do_stop("can0")) {
                perror("Stop␣CAN␣device:␣");
                return -1;
        }

        return 0;
}
```

Listing 6: Debian Multiarch Example

In order to build this program invoke:

```
arm-linux-gnueabihf-gcc -o test test.c -lsocketcan
```

Then copy resulting binary to the OnRISC device. On the device execute `ldd` and you'll see linked libraries including `libsocketcan`:

```
# ldd test
libsocketcan.so.2 => /usr/lib/arm-linux-gnueabihf/libsocketcan.so.2 (0xb6f1b000)
libc.so.6 => /lib/arm-linux-gnueabihf/libc.so.6 (0xb6e2c000)
/lib/ld-linux-armhf.so.3 (0xb6f31000)
```

See https://wiki.debian.org/CrossToolchains for more information about multiarch support in Debian.

Alternatively you can use Buildroot (refer to Section 9) as your development environment. This way you can develop applications with various dependencies like Qt, Boost, Gtk etc. directly on your development host.

6.2. Linux Kernel

6.2.1. Getting Source Code

Baltos Kernel config, patches and Device Tree[30] source files are part of Buildroot's BSP described in Section 9. To get the source code first build Buildroot's default image, then you can change kernel configuration invoking `make linux-menuconfig` from Buildroot's folder.

Buildroot creates a so called FIT image (`*.itb`). This file combines both zImage and DTB blobs in one file. Please refer to these slides[31] and `board/vscom/baltos/custom-FIT.sh` for more details. To install the kernel copy `kernel-fit.itb` to the FAT partition of your SD/microSD-card. See Section 6.2.2 for kernel modules handling. You are advised to always install `kernel-fit.itb` together with kernel modules from the same build to avoid versioning issues.

6.2.2. Install Kernel Modules

Compilation Outside Buildroot Linux kernel provides means to create installation package. The most portable packaging type is a tar package. After kernel compilation invoke:

```
make tar-pkg
```

This will create `linux-x.y.z.tar` file in the root of your kernel tree.

Copy this package to the OnRISC device via `scp` and extract:

```
scp linux-x.y.z.tar root@192.168.254.254:/tmp
```

On the device perform:

```
tar xf /tmp/linux-x.y.z.tar -C /
```

You'll now get `/lib/modules/x.y.z` folder created. Remove unneeded files from `/boot` like `System.map-x.y.z` and `vmlinux-x.y.z` to save space.

Compilation Using Buildroot Buildroot's `baltos_defconfig` automatically invokes `modules.sh`. This script creates tar file and copies it to `output/images/modules.tar`.

6.3. Bootloader

6.3.1. Baltos

Baltos uses U-Boot. Buildroot BSP provides a patch[32], that adds support for Baltos devices. In order to modify U-Boot (change NAND partition table etc.) you need to make following steps:

1. download U-Boot version, that is specified in `baltos_defconfig`
2. apply related patch from `board/vscom/u-boot-patches/`
3. add an entry to `local.mk` pointing to U-Boot's folder

[30] http://elinux.org/Device_Tree
[31] http://elinux.org/images/f/f4/Elc2013_Fernandes.pdf
[32] `board/vscom/u-boot-patches/2014.07/uboot-0001-Add-support-for-Baltos-system.patch`

4. execute `make` in Buildroot's folder

You'll need following files after compilation process:

- `output/images/MLO`
- `output/images/u-boot.img`

If you're booting from SD card, then just copy them to the first partition (FAT). In the case of NAND burn them to following partitions:

- `MLO` to `/dev/mtdblock0`
- `u-boot.img` to `/dev/mtdblock4`

6.4. Other Programming Languages Than C/C++

You can also use other programming languages like Python, Perl etc. The system image already provides Python 2.7.x and Perl 5.20.2. Basically you can install whatever programming language Debian 8 itself provides. To install for example Java invoke:

```
apt-get install openjdk-7-jdk
```

6.5. Recommended Books

A list of books about Linux programming/usage and related topics:

- "The Debian Administrator's Handbook" by Raphaël Hertzog, Roland Mas (http://debian-handbook.info/)
- "Building Embedded Linux Systems, 2nd Edition " by Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, Philippe Gerum
- "Linux Device Drivers, Third Edition" by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman (http://lwn.net/Kernel/LDD3/)
- "C: The Complete Reference" by Herbert Schildt
- "Dive Into Python 3" (http://getpython3.com/diveintopython3/)
- "Pro Git" by Scott Chacon (http://git-scm.com/book)

7. OnRISC Hardware API

7.1. libonrisc

Such hardware as digital I/O, serial interfaces, USB OTG will be controlled through sysfs GPIO entries. To simplify this task `libonrisc` provides a convenient access to these hardware from user space. Following features are implemented:

- get system's hardware info like serial number. MACs etc.
- control LEDs
- control UART's RS232/422/485 driver
- control digital I/O
- read DIP switch
- control mPCIe slot

The library is hosted on GitHub (https://github.com/visionsystemsgmbh/libonrisc) and is included into Debian image and is also part of Buildroot BSP. API documentation can be found in both `README.md` as also `include/onrisc.h`.

`libonrisc` provides a command line tool called `onrisctool` (see Section B). It uses most of the API routines provided by the library and thus its source code is a good starting point to understand API programming. In addition `examples` folder[33] provides dedicated source code samples for different API calls.

[33] https://github.com/visionsystemsgmbh/libonrisc/tree/master/examples

7.2. Digital I/O

Digital I/O is made via TCA6416A I²C I/O expander. GPIOs will be exported via sysfs. See Kernel documentation[34].

For testing purposes one can either export and manipulate I/Os via sysfs directly or use `onrisctool` to control digital I/O (refer to Section B.2). For software development `libonrisc` provides a convenient API (refer to Section 7.1).

7.2.1. Baltos iR 5221/3220

Baltos iR 5221/3220 provides 4 inputs (496-499) and 4 outputs (500 - 503). Inputs and outputs have fixed direction, that cannot be changed. `/sys/class/gpio` will have following entries for 8 GPIOs:

```
gpio496 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio496
gpio497 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio497
gpio498 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio498
gpio499 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio499
gpio500 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio500
gpio501 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio501
gpio502 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio502
gpio503 -> ../../devices/ocp.3/4802a000.i2c/i2c-1/1-0020/gpio/gpio503
```

7.2.2. Baltos 1080

Baltos 1080 provides 8 inputs (496-503) and 8 outputs (504-511). Inputs and outputs have fixed direction, that cannot be changed.

[34] https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

7.3. mPCIe On/Off Switching

In Baltos devices mPCIe slot can be switched on/off in order to enable/disable or reset the installed mPCIe card. Switching can be made either via `onrisctool` or via `libonrisc`. For `onrisctool` please refer to Section B.6. For `libonrisc` usage see a programming example[35] on GitHub.

Please note, that cutting USB device's power is not a standard operation, so please disable the device and unload its driver before switching mPCIe off to avoid unexpected Linux kernel behaviour.

Baltos systems with U-Boot Version "U-Boot 2014.07 (May 11 2016 - 15:38:41)" and Debian image with Kernel 3.18.32 and newer will enable mPCIe already in bootloader, so that depending on modem's firmware a modem is already available when Kernel performs USB device enumeration. Bootloader on older systems can be easily updated using our latest Debian image:

1. boot into Debian
2. invoke `update_bootloader.sh`

You'll get following output:

```
Updating bootloader
Bootloader updated successfully
```

7.4. Serial Interfaces

Depending on the model used OnRISC devices provide serial interfaces based on different UART types. Each UART type has it's own handling. Below you'll find a table describing devices and used UARTs.

Device	UART
Baltos iR 5221/3220/2110, OnRISC 113/213	16C750 (TI OMAP)
Baltos 1080, OnRISC 413/813	FT4232H (FTDI)
VS-860	FT2232D (FTDI)

Table 9: Used UARTs

7.4.1. RS Mode Switching

Two methods are provided to switch between RS232/RS422/RS485 modes (termination inclusive): via software and via DIP-switch. Each port has its own DIP-switch named SW1 and SW2. Table 10 shows possible DIP-switch settings.

Software configuration can be done either via invoking `onrisctool` or using `libonrisc` routines. Below you'll find some `onrisctool` usage examples:

- `onrisctool -p 1 -t rs232` (set first serial port to RS232 mode)
- `onrisctool -p 2 -t rs422` (set second serial port to RS422 mode without termination)
- `onrisctool -p 2 -r -t rs422` (set second serial port to RS422 mode with termination)

[35] https://github.com/visionsystemsgmbh/libonrisc/blob/master/examples/mpcie.c

S1	S2	S3	S4	Mode
off	off	x	x	RS-232 Loopback Test
on	off	x	x	RS-232 MODE
on	on	on	off	RS-422 MODE
on	on	on	on	RS-422 MODE RxD +/- with 120 ohm Term
on	on	off	off	RS-485 Full Duplex Mode
on	on	off	on	RS-485 Full Duplex Mode RxD +/- with 120 ohm Term
off	on	off	off	RS-485 Half Duplex Mode without Echo
off	on	off	on	RS-485 Half Duplex Mode without Echo RxD +/- with 120 ohm Term

Table 10: Serial Modes (hardware switching)

- `onrisctool -p 1 -r -t rs485-hd` (set first serial port to RS485 half-duplex mode with termination)

For `libonrisc` usage see a programming example[36] on GitHub.

Baltos 1080 has RS232 only fixed serial ports, so `onrisctool` will report error, if you try to switch RS modes.

7.4.2. FTDI Based UARTs

FTDI based UARTs have following device naming scheme: `/dev/ttyUSBx`. If the system has two UARTs, they appear in Linux as `/dev/ttyUSB0` and `/dev/ttyUSB1`.

7.4.3. 16C750 Based UARTs

16C750 based UARTs have following device naming scheme: `/dev/ttyOx` (the last letter is 'O' because it is OMAP's UART). If the system has two UARTs, they appear in Linux as `/dev/ttyO1` and `/dev/ttyO2`. In this case the numbering begins with '1' because `/dev/ttyO0` is a console device. These UARTs have also special symlinks:

```
/dev/ttyVS0 -> ttyO1
/dev/ttyVS1 -> ttyO2
```

RS modes will be switched the same way as described in Section 7.4.1. Example below shows, how to configure RS485 mode with termination from software (Please note, that you'll need `libonrisc` installed in order to compile and link this example). To build this example in Debian on OnRISC execute following command:

```
gcc -o rs485 rs485.c -l onrisc
```

`-l onrisc` tells linker to link against `libonrisc`.

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/serial.h>
#include <sys/types.h>
#include <sys/stat.h>
```

[36] https://github.com/visionsystemsgmbh/libonrisc/blob/master/examples/rs485.c

```
#include <fcntl.h>
#include <string.h>
#include <termios.h>
#include <onrisc.h>

#define TEST_BAUDRATE    B9600

int main(int argc, char **argv)
{
        int rc = EXIT_FAILURE;
        int fd = -1, ret;
        struct termios ser_termios;
        onrisc_uart_mode_t mode;
        char buf[32];

        if (argc != 3) {
                printf("Wrong parameter count. \nUsage:\nrs485 device
                   port-number\nExmaple:\nrs485 /dev/ttyO1 1\n");
                goto error;
        }

        if (onrisc_init(NULL) == EXIT_FAILURE) {
                perror("init libonrisc: ");
                goto error;
        }

        mode.rs_mode = TYPE_RS485_HD;
        mode.termination = 1;

        if (onrisc_set_uart_mode(atoi(argv[2]), &mode) == EXIT_FAILURE)
            {
                perror("failed to set serial port mode: ");
                goto error;
        }

        /* open serial port */
        fd = open(argv[1], O_RDWR);
        if (fd < 0) {
                perror("open: ");
                goto error;
        }

        ret = tcgetattr(fd, &ser_termios);
        if (ret < 0) {
                perror("Getting attributes");
                goto error;
        }
        cfmakeraw(&ser_termios);
```

```
        /* configure local flags */
        ser_termios.c_lflag = 0;

        /* configure input flags */
        ser_termios.c_iflag = IGNPAR;

        /* configure output flags */
        ser_termios.c_oflag = 0;

        /* configure control flags */
        ser_termios.c_cflag = CS8 | CLOCAL | CREAD;

        ser_termios.c_cc[VMIN] = 0;
        ser_termios.c_cc[VTIME] = 0;

        ret = cfsetispeed(&ser_termios, TEST_BAUDRATE);
        ret = cfsetospeed(&ser_termios, TEST_BAUDRATE);
        ret = tcsetattr(fd, TCSANOW, &ser_termios);
        if (ret < 0) {
                perror("Setting attributes");
                goto error;
        }

        /* send test string */
        sprintf(buf, "hello");
        ret = write(fd, buf, strlen(buf));

        rc = EXIT_SUCCESS;
error:
        if (fd > 0) {
                close(fd);
        }

        return rc;
}
```

Listing 7: RS485 Half-Duplex with Termination

7.5. CAN

SocketCAN provides access to CAN controller on all CAN capable OnRISC devices. SocketCAN[37] is a set of open source CAN drivers and a networking stack contributed by Volkswagen Research to the Linux kernel. Formerly known as Low Level CAN Framework (LLCF).

7.5.1. CAN Interface Configuration

Special script `/etc/init.d/can_if`[38] is provided to setup `can0` interface. It is preconfigured to use `can0` at 1000000b/s. These settings are stored in the `CAN_IF` variable:

```
CAN_IF="can0@1000000,200"
```

To start the CAN interface issue:

```
/etc/init.d/can_if start
```

to stop CAN interface issue:

```
/etc/init.d/can_if stop
```

Further information regarding programming and configuration can be found on the SocketCAN's project site.

7.5.2. CAN Usage Examples

To send a CAN frame execute:

```
cansend can0 123#1234
```

To receive CAN frames execute:

```
candump can0
```

7.5.3. CANopen

It is possible to use CANopen[39] library from CanFestival[40] or any other open source or proprietary CANopen stack, that has SocketCAN support.

7.6. I²C

I²C[41] (Inter-Integrated Circuit) is a multi-master serial computer bus. In the OnRISC integrated I²C controller is already supported by the mainline kernel. A good starting point on how to use I²C is Linux kernel documentation[42]. `i2c-tools`[43] project provides user space utilities to work with I²C.

[37] http://elinux.org/CAN_Bus
[38] this script is maintained in this git repository: `https://github.com/linux-can/can-misc`
[39] www.can-cia.org
[40] www.canfestival.org
[41] http://en.wikipedia.org/wiki/I2C
[42] https://www.kernel.org/doc/Documentation/i2c/dev-interface
[43] http://www.lm-sensors.org/wiki/I2CTools

7.7. Watchdog Timer

The OnRISC provides a watchdog timer (WDT). The access to the WDT occurs via `/dev/watchdog` device. After starting the WDT it should be turned off or reloaded after the critical part otherwise the system will reboot. Table 11 shows the most important IOCTLs to control WDT.

IOCTL	Description
`WDIOC_SETTIMEOUT`	set timeout and start the timer
`WDIOC_GETTIMEOUT`	get timeout
`WDIOC_SETOPTIONS`	enable (`WDIOS_ENABLECARD`) disable (`WDIOS_DISABLECARD`) the timer
`WDIOC_KEEPALIVE`	reload the timer

Table 11: Watchdog Timer IOCTLs

The WDT driver provides a special option called "Disable watchdog shutdown on close" to prevent stopping the timer on WDT descriptor close (see Figure 4). This is a compile-time option. It is deactivated by default, so you have to build your own kernel if you need it.

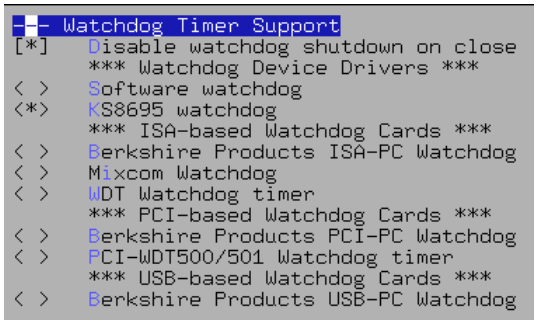

Figure 4: Watchdog Timer Support

See https://www.kernel.org/doc/Documentation/watchdog/src/ for WDT usage example.

7.8. Read Hardware Parameters like MAC Address, Serial Number etc.

Use `onrisctool -s` and you'll get a list of all parameters stored in EEPROM. For software development `libonrisc` (refer to Section 7.1) should be used to get hardware parameters.

7.9. Built-in Touchscreen Calibration (VS-860 Only)

You can choose between two calibration methods:

- tslib[44] - suitable for Qt Embedded applications
- X11 evdev driver - as the name says suitable for X11

Tslib package provides ts_calibrate utility, to calibrate the touchscreen. New calibration values will be written to `/etc/pointercal`. Tslib requires following environment variables to use the touchscreen:

```
TSLIB_CALIBFILE=/etc/pointercal
TSLIB_CONFFILE=/etc/ts.conf
TSLIB_TSDEVICE=/dev/input/event1
```

Hwtest-qt utility provides special Python script (scripts/input_dev.py), that reads /proc/bus/input/devices and configures `TSLIB_TSDEVICE` together with Qt Embedded related environment variables.

X11 evdev driver will be calibrated via xinput_calibrator (must be started from X11 session). The calibration values will be stored in `/usr/share/X11/xorg.conf.d/10-evdev.conf`. Example:

```
Section "InputClass"
Identifier "evdev touchscreen catchall"
MatchIsTouchscreen "on"
MatchDevicePath "/dev/input/event*"
Option "Calibration" "28 999 988 41"
Driver "evdev"
EndSection
```

[44] https://github.com/kergoth/tslib

8. vsdebootstrap

Debian provides tools to create custom root file system directly on the host. One of these tools is `debootstrap`[45]. Our system image is created with a help of scripts wrapped around `debootstrap`. You can find the scripts and corresponding documentation on our GitHub account: vsdebootstrap. `vsdebootstrap` not only creates root file system with OSS packages like `openvpn`, `ssh` etc., but also installs our packages like `libonrisc` and `hwtest-qt`. In addition it provides facilities to embed your own packages and configuration files into resulting root file system. By default `vsdebootstrap` creates an image for the current stable Debian version.

9. Buildroot

Buildroot[46] is a set of Makefiles and patches that allows you to easily generate a cross-compilation toolchain, a root filesystem and a Linux kernel image for your target. Buildroot can be used for one, two or all of these options, independently. The benefits are:

- you don't have to supply the toolchain it will be built automatically
- you can choose between various file systems for your target image
- you can build a very small and quick booting system that suits your needs
- drivers compiled as modules will be automatically installed on your file system image
- many more

Free Electrons provides very informative slides[47] related to Buildroot usage. We recommend to go through them before you begin using our BSP.

9.1. Build Host Requirements

Before building your root file system please check BR's software requirements[48]. It is also recommended, but not mandatory to build BR on a machine with more than 1GB RAM. You'll also need at least 6GB of free disk space in order to download and build needed packages.

9.2. Downloading

`README.md` in `onrisc_br_bsp` specifies a minimal required Buildroot version, so you can either use this tag or just clone master branch as shown below. Buildroot provides BR2_EXTERNAL[49] feature, that lets you keep your configuration and packages outside of Buildroot itself. To get our BSP execute following steps:

1. git clone https://github.com/visionsystemsgmbh/onrisc_br_bsp.git
2. git clone git://git.buildroot.net/buildroot or git clone http://git.buildroot.net/git/buildroot.git (if you're behind a firewall)

[45] https://wiki.debian.org/Debootstrap
[46] http://buildroot.org/
[47] http://free-electrons.com/doc/training/buildroot/buildroot-slides.pdf
[48] http://nightly.buildroot.org/manual.html#requirement
[49] http://nightly.buildroot.org/manual.html#outside-br-custom

3. cd buildroot
4. make BR2_EXTERNAL=../onrisc_br_bsp/ help

9.3. BSP Structure

- `board/` - this folder provides actual BSP, i.e. kernel patches, kernel config etc.
- `Config.in` - Buildroot's local package config file
- `configs/` - Buildroot's defconfigs per device like baltos_defconfig etc.
- `external.mk` - Buildroot's local package config file
- `local.mk`[50] - file needed to override package source directory
- `package/` - local packages like libonrisc and packages created by customer
- `README.md` - BSP documentation

9.4. Building the Image

After checking out the repository you can build the default image by issuing the following command:

```
make baltos_defconfig && make
```

Buildroot will start to automatically download needed packages and compile them. At the end of this procedure you'll find built images under:

```
output/images
```

The rootfs image can now be burned on to the target media.

9.5. Copying the Created Image to the System

9.5.1. SD/microSD-card

Copy Genimage Created Image `onrisc_br_bsp` provides a script `post-image.sh`[51], that creates a ready-to-burn image for the SD/microSD-card including both FAT and ext4 partitions. After the build process is finished you'll find following file under `output/images`:

```
sdcard.img
```

This file can then be directly burned via `dd` or Win32 Disk Imager. See Section D.1 for further details.

[50] http://nightly.buildroot.org/manual.html#_using_buildroot_during_development
[51] https://github.com/visionsystemsgmbh/onrisc_br_bsp/blob/master/board/vscom/baltos/post-image.sh

Copy Separate Images Manually On your SD/microSD-card you need to create two partitions as described in Section 4.2.1. Copy following files from output/images to the FAT partition:

- `MLO` - SPL
- `u-boot.img` - U-Boot
- `kernel-fit.itb` - zImage + DTB as FIT binary

After this copy `onrisc_br_bsp/board/vscom/baltos/uEnv.txt` to the same partition.

Root file system is packaged as a tarball `output/images/rootfs.tar.bz2`. All you need to do is to extract it to the second partition. Following steps will be needed on Debian/Ubuntu to burn rootfs assuming, that SD/microSD-card is assigned to `/dev/sdd`:

```
sudo mount /dev/sdd2 /mnt
sudo rm -fr /mnt/*
sudo tar xjf output/images/rootfs.tar.bz2 -C /mnt/
sudo umount /mnt
```

9.5.2. NAND

Copy `MLO`, `u-boot.img`, `kernel-fit.itb` and `rootfs.tar.bz2` to your SD/microSD-card, for example to `/root/`. Boot from this SD/microSD-card and perform following commands:

1. `cd /root/`
2. `cat MLO > /dev/mtdblock0`
3. `cat u-boot.img > /dev/mtdblock4`
4. `ubiformat -y /dev/mtd5`
5. `ubiattach -p /dev/mtd5`
6. `ubimkvol /dev/ubi0 -N kernel -s 56MiB`
7. `mount -t ubifs ubi0:kernel /mnt`
8. `cp kernel-fit.itb /mnt`
9. `umount /mnt`
10. `ubimkvol /dev/ubi0 -N rootfs -s 180MiB`
11. `mount -t ubifs ubi0:rootfs /mnt`
12. `tar xjf rootfs.tar.bz2 -C /mnt`
13. `umount /mnt`

Please refer to this article[52] about flashing UBIFS images for more details.

[52] http://free-electrons.com/blog/creating-flashing-ubi-ubifs-images/

9.6. Customizing the Image

Buildroot uses Kconfig[53] subsystem to manage the process of configuration just like Linux kernel does. Following bigger parts can be configured:

- `make menuconfig` - configures Buildroot: toolchain, rootfs, packages
- `make busybox-menuconfig` - configures BusyBox[54]
- `make uclibc-menuconfig` - configures uClibc[55]
- `make linux-menuconfig` - configures Kernel

First place to look for choosing new packages is BusyBox configuration. If the package is not available there, then it can be found in Buildroot itself, if it was included into its repository.

Buildroot provides file system overlay facility in order to store files, that should be added to the rootfs like configuration files (/etc/network/interfaces etc.), SSH keys etc. Aside from this you can specify own scripts, that will be invoked before or after root file system is packaged (`BR2_ROOTFS_POST_BUILD_SCRIPT` and `BR2_ROOTFS_POST_IMAGE_SCRIPT`).

9.7. Compiling Your Own Software

There are following ways on how to get your software to be build and installed on the system using Buildroot:

- adding your software to the `onrisc_br_bsp/package` folder (recommended)
- using generated toolchain to compile your software and then manually copying it to `output/target`

The first way is described here Add packages and showed in the example below. The second way is described here Using toolchain. See this article on how to override source directory Source directory override.

You can also use Buildroot together with Eclipse to develop your application Eclipse integration.

Example This sample project will show, how to create a CMake package with for example libsocketcan dependency in BR. First create sample project folder:

```
mkdir /home/user/myprog
```

Then create `test.c` from Listing 6 and `CMakeLists.txt` with following content:

```
project(myprog C)
add_executable(test test.c)
target_link_libraries(test socketcan)
install(TARGETS test RUNTIME DESTINATION bin)
```

[53] en.wikipedia.org/wiki/Kconfig
[54] www.busybox.net
[55] www.uclibc.org

Now you need to create package recipe in `onric_br_bsp`:

```
mkdir package/myprog
touch package/myprog/Config.in
touch package/myprog/myprog.mk
```

`package/myprog/Config.in` will have following content:

```
config BR2_PACKAGE_MYPROG
        bool "myprog"
        select BR2_PACKAGE_LIBSOCKETCAN
        help
          Package example
```

`package/myprog/myprog.mk`:

```
################################################################################
#
# myprog
#
################################################################################
MYPROG_VERSION = 1.0.0
MYPROG_SITE = git://git.mygitsrv.com/myprog.git
MYPROG_DEPENDENCIES = libsocketcan
$(eval $(cmake-package))
```

`package/myprog/Config.in` must be included in central `Config.in`:

```
source "$BR2_EXTERNAL/package/libonrisc/Config.in"
source "$BR2_EXTERNAL/package/myprog/Config.in"
```

As `MYPROG_SITE` is not available and we want Buildroot to compile the working copy of myprog in `/home/user/myprog`, `local.mk` must be edited to point to myprog's folder:

```
MYPROG_OVERRIDE_SRCDIR=/home/user/myprog
```

You can now select `myprog` via `make menuconfig` and entering "User-provided options". You'll see `myprog` next to `libonrisc`. Select `myprog` and invoke `make myprog-rebuild`. This will rsync `/home/user/myprog` into `output/build/myprog-custom` and build/install it. So every time you make changes to `test.c`, just invoke `make myprog-rebuild` to rebuild it.

9.8. Setup SSH Server

SSH server will be activated via the BR2_PACKAGE_OPENSSH variable. After copying the new rootfs image to the media, you'll see RSA/DSA keys will be created. By default password is required to login via SSH. You can configure the system in the following ways:

- define root password via `passwd root`
- change `sshd` behavior to accept empty passwords (set `PermitEmptyPasswords` to `yes` in `/etc/sshd_config`)

9.9. Getting Help

If you encounter any non-kernel related problems, you should get in contact with Buildroot community. As usual it is recommended to read the mailing list archives before asking questions on the mailing list.

10. Yocto

The Yocto Project[56] is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. This section provides step-by-step instructions on how to add Baltos BSP layer to your project. It is not meant as a full Yocto tutorial. You'll find links to particular sections of the Yocto documentation describing variables and files used in this mini tutorial.

10.1. Downloading

Layers are the main Yocto concept and consist of a set of recipes matching a common purpose. To build an image having a kernel and `libonrisc` for Baltos you'll need to add layer `meta-baltos` to your `bblayers.conf` file. Due to `libonrisc` dependencies you'll have to use Yocto 2.1 (krogoth) or newer. If you build Yocto for the first time, you'll need to perform following steps:

1. `mkdir /home/user/yocto`
2. `cd /home/user/yocto`
3. `git clone -b krogoth git://git.yoctoproject.org/poky.git`
4. `cd poky`
5. `git clone git://git.openembedded.org/meta-openembedded`
6. `git clone https://github.com/visionsystemsgmbh/meta-baltos.git`
7. `source oe-init-build-env`

10.2. Build Configuration

After these steps you'll have a basic setup of all needed repositories and are ready to configure your rootfs image. First of all you'll need to add cloned layers to `conf/bblayers.conf`[57]. Open this file and add baltos and openembedded layers so it looks like this:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
BBLAYERS ?= " \
/home/user/yocto/poky/meta \
/home/user/yocto/poky/meta-poky \
/home/user/yocto/poky/meta-yocto-bsp \
/home/user/yocto/poky/meta-baltos \
/home/user/yocto/poky/meta-openembedded/meta-oe \
"
BBLAYERS_NON_REMOVABLE ?= " \
/home/user/yocto/poky/meta \
```

[56] https://www.yoctoproject.org/
[57] http://www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#structure-build-conf-bblayers.conf

```
/home/user/yocto/poky/meta-poky \
"
```

Add following lines to `conf/local.conf`[58] to include Baltos BSP into the build process:

- `MACHINE ??= "baltos"`[59]
- `IMAGE_INSTALL_append = " libonrisc"`[60]

Invoking `bitbake core-image-base` would start the build process. At the end you'll find all needed images under `tmp/deploy/images/baltos`[61]:

- `tmp/deploy/images/baltos/kernel-fit.itb` - zImage + DTB as FIT binary
- `tmp/deploy/images/baltos/core-image-base-baltos.tar.bz2` - symbolic link to the rootfs
- `tmp/deploy/images/baltos/MLO` - symbolic link to SPL
- `tmp/deploy/images/baltos/u-boot-img` - symbolic link to U-Boot

Use steps from Copying the Created Image to the System in order to copy these files to your boot medium.

10.3. Meta-baltos Structure

Baltos BSP layer provides following files/recipes:

- `conf/machine/baltos.conf` - initial configuration for kernel and u-boot
- `recipes-bsp` - recipes for libonrisc and u-boot as well as u-boot patches
- `recipes-kernel` - kernel recipe, defconfig and patches

[58] http://www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#structure-build-conf-local.conf
[59] http://www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#var-MACHINE
[60] http://www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#var-IMAGE_INSTALL
[61] http://www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#structure-build-tmp-deploy-images

A. Debian Maintenance Notes

A.1. Debian Package Management

Debian uses following utilities for the package management[62] :

- dpkg - the main package management program.
- APT - the Advanced Package Tool. It provides the `apt-get` program. `apt-get` allows a simple way to retrieve and install packages from multiple sources using the command line. Unlike `dpkg`, `apt-get` does not understand .deb files, it works with the packages proper name and can only install .deb archives from a source specified in `/etc/apt/sources.list`. `apt-get` will call `dpkg` directly after downloading the .deb archives from the configured sources.
- aptitude - a package manager for Debian GNU/Linux systems that provides a frontend to the apt package management infrastructure. `aptitude` is a text-based interface using the curses library, it can be used to perform management tasks in a fast and easy way.

To find a package execute:

`apt-cache search` *`pkg name`*

To view info such as version, dependencies, installed size etc. execute:

`apt-cache show` *`pkg name`*

To install packages execute:

`apt-get install` *`pkg1 pkg2 ...`* (su rights needed)

To update the list of package known by your system execute:

`apt-get update` (su rights needed)

To upgrade all the packages on your system

`apt-get upgrade` (su rights needed)

To remove packages from your system execute:

`apt-get remove` *`pkg1 pkg2 ...`* (su rights needed)

To install a package that is not contained in the repository download the *.deb file and execute:

`dpkg -i` *`pkg file name`* (su rights needed)

[62] for detailed information visit http://www.debian.org/doc/FAQ/ch-pkgtools.en.html

B. onrisctool

OnRISC configuration utility provides following features to configure OMAP3 based devices:

- configure serial interfaces (RS232, RS422 etc.)
- configure digital I/O
- configure LEDs
- get EEPROM info
- set MACs from EEPROM
- enable/disable mPCIe slot

B.1. Configure Serial Interfaces

Baltos and VS-860 provide two UARTs working in RS232/RS422/RS485 modes. Invoking `onrisctool -j` will show which mode each port is configured to:

```
# onrisctool -j
Port 1:  mode:  rs232 termination:  off source:  DIP
Port 2:  mode:  rs232 termination:  off source:  DIP
```

As one can see both ports are configured in RS232 mode through DIP-switches. To switch the first port into RS422 mode execute:

```
onrisctool -t rs422 -p 1
```

All possible modes are show in the Table 12. Termination can be enabled via `-r` parameter.

Mode	Description
dip	DIP-switch settings are valid
loop	loopback mode
rs232	RS232
rs422	RS422
rs485-fd	RS485 full duplex
rs485-hd	RS485 half duplex

Table 12: Serial Modes (Software switching)

B.2. Configuring Digital I/O

Baltos iR 5221/3220 provides 4 digital inputs and 4 digital outputs, Baltos 1080 provides 8 digital inputs and 8 digital outputs. With `onrisctool` you can read and modify data (see Table 13 on page 55).

Following example for Baltos iR 5221/3220 turns pins 0 and 2 to high and clears pin 1:

```
onrisctool -a 0x70 -b 0x50
```

The same example but for Baltos 1080:

Parameter	Description
`-a`	Bit mask. Defines, what bits will be affected via the data
`-b`	Data bits, that will affect only the bits defined by `-a` parameter
`-g`	Show current state

Table 13: GPIO Parameters

```
onrisctool -a 0x0700 -b 0x0500
```

Show current state:

```
# onrisctool -g
0x00000505
```

Figure 5 on page 55 shows, how mask and value are mapped to input/output connector for Baltos iR 5221/3220. In this example IN0 is connected to OUT0 and OUT0 has high level. As one can see 4 bytes are reserved, so that devices with more digital inputs/outputs can be supported (for example Baltos 1080, that uses two bytes), but for Baltos iR 5221/3220 only the least significant byte is important. Baltos 1080 uses least significant byte for inputs and the next byte for outputs.

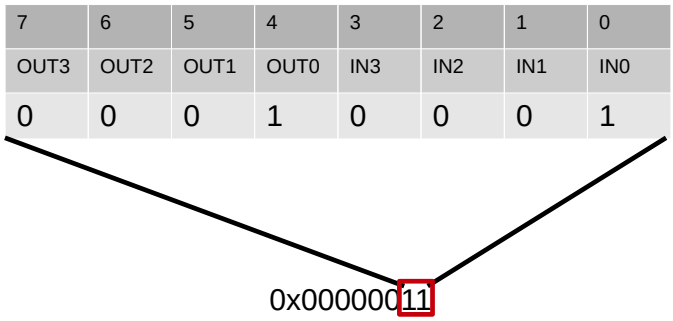

Figure 5: Baltos: Digital I/O Mapping

B.3. Configuring LEDs

Baltos devices provide up to 3 user configurable LEDs. LEDs will be configured via `-l` parameter. The call consists of an LED name (see Table 14 on page 55) and desired function (see Table 15 on page 56). In the example below power LED will be turned off:

```
onrisctool -l pwr:0
```

Name	Description	Color
pwr	power LED	red
app	application LED	green
wln	WLAN LED	blue

Table 14: LED Names

Function	Description
0	turn LED off
1	turn LED on
2	blink with different duty cycles for some seconds
3	get current LED state (on/off)

Table 15: LED Functions

B.4. Get EEPROM Info

`onrisctool -s` shows hardware related data like MACs, serial number etc. stored in EEPROM:

```
# onrisctool -s
Hardware Parameters
===================
Model:  210
HW Revision:  1.2
Serial Number:  550100239
Production date:  20.06.2015
MAC1:  746a8f0001f5
MAC2:  746a8f0001f6
MAC3:  746a8f0001f7
```

B.5. Setting LAN MACs from EEPROM

You can use in EEPROM stored MACs for internal LAN interfaces:

`onrisctool -m`

Executing this command will set MAC1 and MAC2 to `eth0` and `eth1/usb0`.

B.6. Controlling mPCIe Slot

The mPCIe slot in Baltos devices can be switched on and off via GPIO pin. This slot is off by default and must be enabled on demand. To do this invoke:

`onrisctool -k 1`

To disable the slot invoke:

`onrisctool -k 0`

To check current state invoke:

`onrisctool -k 3`

This would output following message, if the slot is on:

```
mPCIe switch:  on
```

C. hwtest-qt

Hwtest-qt provides various hardware tests in both console and graphical environment. To get GUI start `ghwtest-qt` instead of `hwtest-qt`. Hwtest-qt can be run using special configuration file.

```
hwtest-qt -qws -c /etc/hwtest.conf
```

If you want to execute the tests in parallel:

```
hwtest-qt -qws -c /etc/hwtest.conf --parallel-exec
```

Please see `hwtest-qt -qws --help` for further options.

C.1. Controller Area Network Test

CAN test requires USB-CAN device attached to one of the USB ports[63]. Connect internal CAN interface with USB-CAN via CAN cable and execute:

```
hwtest-qt -qws --test-can --cani=slcan0 --cano=can0
```

C.2. UART Test

Connect both internal serial ports via null-mode, cable and execute:

```
hwtest-qt -qws --test-serial --seri=/dev/ttyO1 --sero=/dev/ttyO2 --seropts="-t rs232"
```

C.3. Network Test

Connect both LAN insteraces via patch cable and execute:

```
hwtest-qt -qws --test-network --neti=eth0 --neto=eth1
```

C.4. RTC Test

```
hwtest-qt -qws --test-rtc
```

C.5. WLAN Test

WLAN test scans for available WLAN hosts (iw wlan0 scan). At least one AP must be running and antenna must be attached.

```
hwtest-qt -qws --test-wlan
```

[63] You'll need a slcan tools to bring USB-CAN up. See VSCAN SocketCAN FAQ

C.6. Bluetooth Test

Bluetooth test needs an active Bluetooth device like headset, Bluetooth dongle etc. to find during the scan.

```
hwtest-qt -qws --test-bt
```

C.7. Disk Test

Insert CFast card and execute:

```
hwtest-qt -qws --test-disk --drive=/dev/sda
```

C.8. Touch Test

If you start ghwtest-qt and you can precisely touch the buttons, then touch controller is functioning properly. You can calibrate it by executing

```
ts_calibrate
```

C.9. Button Test

For button test the user will be prompted to press buttons on the front panel from left to right, skipping the first button (power button).

```
hwtest-qt -qws --test-btn
```

C.10. Audio Test

Insert earphones into the left connector and execute:

```
hwtest-qt -qws --test-audio
```

You'll hear a woman voice saying "Front left"

D. Managing System Images

D.1. Flashing System Images

D.1.1. Windows

Win32 Disk Imager[64] was developed to make/copy images from/to the flash drives connected to a Windows host (see Figure 6).

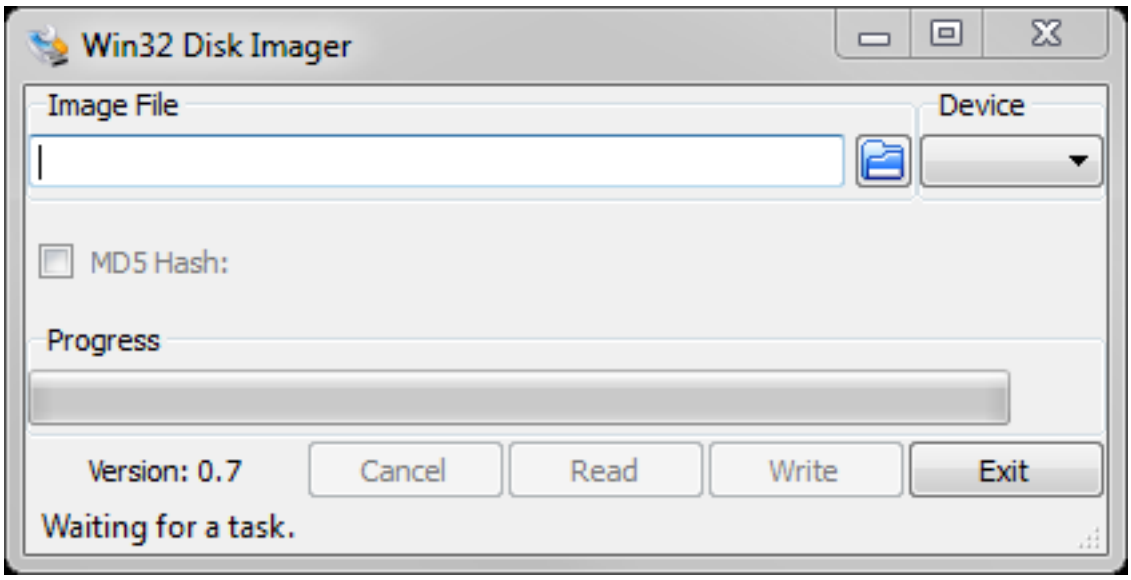

Figure 6: Win32 Disk Imager

[64] http://sourceforge.net/projects/win32diskimager/

D.1.2. Linux

From Host PC To copy a system image to a SD/microSD-card using your host PC just download the required system image from our FTP server and extract the *.img file (for example to `/home/user/`).

If USB card reader is used, the SD/microSD-card could be found under `/dev/sdx` devices (for example `/dev/sda`, `/dev/sdb` etc). Please note, that the system images are complete images of a SD/microSD-card not of the single partition and must be applied to the device like `/dev/sda` and not `/dev/sda1`.

For the example below it will be assumed that the SD/microSD-card is assigned to `/dev/sda` device and the system image is `20140805_debian_7_baltos.img`:

```
su
dd if=/home/user/20140805_debian_7_baltos.img of=/dev/sda bs=4096
```

To create an image from the SD/microSD-card execute following:

```
dd if=/dev/sda of=/home/user/image.img
```

D.2. Working with Partitions

It is also possible to write/extract only one partition at file system level. The advantage: you can write this partition even to a smaller medium. Please refer to this project FSArchiver

E. Frequently Asked Questions (FAQ)

There is dedicated website http://faq.visionsystems.de to handle the FAQ concerning OnRISC and other products provided by VScom. The customer question will be posted to the support team for approval and if approved appears in the corresponding category.