NetModule OEM Linux Distribution Documentation

Release 0.4.3

NetModule

Mar 05, 2021

About

1 Contributing **3**

2 Release Notes **5**

3 [1.2.1] - 2021.03.05 **7**
- 3.1 Added . . . 7
- 3.2 Changed . . . 7
- 3.3 Fixed . . . 7
- 3.4 Known Issues . . . 7

4 Supported devices **9**
- 4.1 NG800 . . . 9
- 4.2 NB800 . . . 9
- 4.3 NB1601 . . . 9
- 4.4 NB1800 . . . 10

5 Key Features **11**
- 5.1 Yocto Project Support . . . 11
- 5.2 NetModule Hardware Support . . . 11
- 5.3 Network Configuration . . . 11
- 5.4 Automotive Grade Firmware Upgrade (OSTree) . . . 11
- 5.5 Package Manager . . . 11

6 Setup your Board **13**
- 6.1 Image Installation . . . 13

7 Build your Image **17**

8 Getting Started: NetModule Linux **19**
- 8.1 NetModule Addons . . . 19
- 8.2 Image Types . . . 19

9 OSTree **21**
- 9.1 Description . . . 21

10 Artifacts for nmhw21 **25**
- 10.1 wic-file . . . 25

11 Artifacts overview **27**
11.1 wic-file . . . 27
11.2 Kickstart/wks-file . . . 27

12 am335x Startup **29**

13 eMMC Contents **31**

14 Startup Sequence **33**

15 Automatic Partitioning **35**
15.1 Recommended settings . . . 35
15.2 Usage . . . 35
15.3 Examples . . . 36
15.4 How to undo the partitioning . . . 36

16 Bluetooth **37**
16.1 Commands . . . 37
16.2 Start and Discover nearby devices . . . 37

17 CAN **39**
17.1 CAN Interfaces . . . 39
17.2 SocketCAN . . . 40
17.3 can-utils . . . 41

18 Chrony **43**
18.1 Usage . . . 43
18.2 More configuration option . . . 44

19 Package Management using DNF **45**
19.1 Setup . . . 45
19.2 Commands . . . 45
19.3 OSTree compatibility . . . 45

20 Ethernet **47**
20.1 Overview . . . 47
20.2 Supported scenarios . . . 47
20.3 Supported hardware . . . 47
20.4 NXP SJA1105TEL switchdev driver . . . 48
20.5 NXP TJA110X PHY driver . . . 49
20.6 User space tools and configuration . . . 50
20.7 Multiple isolated interfaces example . . . 51
20.8 References . . . 52

21 Updating firmware **53**
21.1 Supported modules . . . 53
21.2 Firmware Location . . . 53
21.3 Preparation . . . 53
21.4 Usage . . . 54

22 GNSS **57**
22.1 gpsd . . . 57
22.2 NEO-M8L . . . 57
22.3 gnss-mgr . . . 57
22.4 Configuring the GNSS Receiver . . . 59
22.5 Save on Shutdown . . . 62

22.6 Firmware update (specific to u-blox NEO-M8L) . . . 62
22.7 Testing . . . 62
22.8 Access gps interface . . . 63

23 GSM **65**

24 IMU **67**
24.1 Polling mode . . . 67
24.2 Buffered mode . . . 68

25 LEDs **71**
25.1 Standard LED behavior . . . 71
25.2 LED: PCB Ind . . . 72
25.3 LED: PCB Status . . . 72
25.4 LED: UI Ind . . . 72
25.5 LED: UI Status . . . 73

26 Linux System Logging **75**
26.1 Accessing Log Files . . . 75
26.2 Storage of the Logs . . . 75
26.3 Configuration Parameters . . . 76
26.4 Maintaining the Logs . . . 76

27 NetModule Linux Networking **79**
27.1 NetworkManager . . . 79

28 Power Management **81**
28.1 Overview . . . 81
28.2 Smart Battery . . . 82

29 Remote GPIO driver **83**
29.1 Configuration . . . 83
29.2 Protocol . . . 84
29.3 Usage . . . 85

30 System Clock / Date Time **87**
30.1 Synchronization . . . 87
30.2 HW Clock Synchronization/Update . . . 87

31 Wi-Fi **89**
31.1 Overview . . . 89
31.2 Client Mode . . . 89
31.3 Access Point Mode . . . 90

32 WWAN **91**
32.1 Overview . . . 91
32.2 Preparation . . . 91
32.3 Usage . . . 91

33 Booting with custom Linux kernel or ramdisk **95**
33.1 Provisioning over tftp . . . 95
33.2 Provisioning over USB . . . 95

34 Create a fitImage **97**
34.1 Steps (nmhw21) . . . 97

35 Indices and tables 99

For more than 15 years NetModule has proven itself as a reliable OEM partner with thousands of installed devices for customers all over the world!

NetModule OEM platforms come with the feature-rich NetModule Router Software or a standard Yocto Linux for customer-specific applications.

Contents:

CHAPTER 1

Contributing

CHAPTER 2

Release Notes

CHAPTER 3

[1.2.1] - 2021.03.05

3.1 Added

- [70484] gnss-mgr: Added config option to disable dead reckoning
- [70564] doc: added warning for worse results with a misconfigured GNSS with DR
- [70360] SSF/bootloader: added detection of reset/start reason
- [58737] HW23: u-boot: Added support for board descriptor

3.2 Changed

- [70602] doc: updated ostree repository reference (upgrad of yocto from warrior to dunfell)
- [67750] HW23: Fixed failing V2X service after software reboot

3.3 Fixed

- [70628] image-minimal: Removed ModemManager
- [70629] image-minimal: Removed wwan-config
- [70346] HW23: u-boot: Set ethernet MAC address based on board descriptor
- [70308] HW23: Fixed CAN3 interface and SPI3 pinmuxing

3.4 Known Issues

- [63349] ostree: python cache files (.pyc) not updated

CHAPTER 4

Supported devices

4.1 NG800

- *Telematic Control Unit*
- Interfaces: BroadR, CAN, Ethernet, Wi-Fi, LTE, GNSS, BT
- Interna interface for user modules: Ethernet, USB, SPI, I2C, GPIO
- CPU: TI am335x, 1000MHz
- BSP: nmhw21

4.2 NB800

- *Customer-specific OEM Router*
- Interfaces: LTE, Wi-Fi, Bluetooth and BLE
- CPU: TI am335x, 600MHz
- BSP: nrhw16, nrhw24

Product Page NB800

4.3 NB1601

- *Ruggedized OEM Router*
- Interfaces: LTE, WiFi, GNSS and 4x Ethernet
- CPU: TI am335x, 600MHz
- BSP: nrhw20

Product Page NB1601

4.4 NB1800

- BSP: nrhw18

CHAPTER 5

Key Features

5.1 Yocto Project Support

5.2 NetModule Hardware Support

5.3 Network Configuration

5.4 Automotive Grade Firmware Upgrade (OSTree)

5.5 Package Manager

CHAPTER 6

Setup your Board

6.1 Image Installation

6.1.1 Over the Network

Note: This procedure will remove all contents on the eMMC. Make sure to backup your data!

Setup a HTTP server

1. Install python3

```
sudo apt install python3
```

2. Set your ip to 192.168.1.254 (If you use a network-manager set your static ip there! This command will not work in this case.)

```
sudo ifconfig <Your ethernet device e.g. eth0> 192.168.1.254
```

3. Make a new folder and enter it

```
sudo mkdir /srv/http
cd /srv/http
```

4. Start the http-server

```
sudo python3 -m http.server --bind 192.168.1.254 80
```

Setup a TFTP server

1. Install the following packages

```
sudo apt-get install xinetd tftpd tftp
```

2. Edit or make a file in: `/etc/xinetd.d/tftp` with the following content:

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = -s /srv/tftp
disable         = no
}
```

3. Create a tftp folder

```
mkdir -p /srv/tftp
```

4. Restart the xinetd service

```
sudo service xinetd restart
```

Instructions

1. Put the WIC file (e.g. *image-am335x-nmhw21.wic*) file on the http-server (*/srv/http*).
2. Put the Minimal Linux fitImage (e.g. *fitImage-am335x-nmhw21.bin*) on the tftp-server (*/srv/tftp*).
3. Verify the */srv* folder:

```
/srv
|-- http
|   |-- image-am335x-nmhw21.wic
|-- tftp
.   |-- fitImage-netmodule-linux-image-minimal-am335x-nmhw21
```

4. Power up the board and press 's' on the serial terminal to stop in u-boot.
5. Validate that you are in u-boot console by typing

```
# env print serverip
serverip=192.168.1.254
```

6. Load kernel binary into ram

```
# tftp $ramdisk_addr_r fitImage-netmodule-linux-image-minimal-am335x-nmhw21
```

7. restrict positioning of initrd ramdisk image

```
# setenv initrd_high 0x84000000
```

8. Set boot args

```
# setenv bootargs root=/dev/ram0 console=ttyS2,115200 ti_cpsw.rx_packet_max=1526
```

9. Boot from ramdisk. Linux will boot to login prompt within < 30s

```
# bootm $ramdisk_addr_r
```

10. Login as *root* (empty password).
11. Disable kernel messages in Linux system

```
echo 1 >  /proc/sys/kernel/printk
```

12. Burn the *.wic* file to the eMMC: (this takes about 5 min)

```
curl http://192.168.1.254/image-am335x-nmhw21.wic | dd of=/dev/mmcblk1 bs=10M && sync
```

13. Reboot your system.

```
reboot
```

6.1.2 From USB Stick

Prepare USB Stick

Format a USB-pendrive to have a with **fat** with a **maximum partition size of 4.0GB** for the first partition.

Use `gnome-disks` or *gparted*‘ for this.

The output of *parted -l* should look like this:

```
Model: Kingston DataTraveler 3.0 (scsi)
Disk /dev/sdb: ...GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End       Size     Type     File system  Flags
1       1049kB  4001MB    4000MB   primary  fat32        lba
```

1. Put the *image-am335x-nmhw21.wic* file on a USB-pendrive.
2. Put the files *image-minimal-am335x-nmhw21.cpio.gz.u-boot, fitImage-am335x-nmhw21.bin* on the USB-pendrive.
3. Your USB-pendrive should no look like this:

```
USB-root
|-- image-am335x-nmhw21.wic
|-- image-minimal-am335x-nmhw21.cpio.gz.u-boot
|-- fitImage-am335x-nmhw21.bin
```

Flash from USB Stick

1. Plug the USB-pendrive into the nmhw21.
2. Power up the board and press ‘s’ on the serial terminal to stop in u-boot.
3. Boot the ramdisk (copy-paste into u-boot terminal):

```
usb reset; fatload usb 0:1 $kernel_addr_r fitImage-am335x-nmhw21.bin; fatload usb 0:1
↪$ramdisk_addr_r image-minimal-am335x-nmhw21.cpio.gz.u-boot; setenv bootargs root=/
↪dev/ram0 console=ttyS2,115200 ti_cpsw.rx_packet_max=1526; bootm $kernel_addr_r
↪$ramdisk_addr_r
```

4. When the the system has booted, log in with user *root*.
5. Mount the USB-pendrive

```
mount /dev/sda1 /mnt
```

6. Burn the *.wic* file to the eMMC: (this takes about 5 min)

```
dd if=/mnt/image-am335x-nmhw21.wic of=/dev/mmcblk1 bs=10M && sync
```

7. Reboot your system

```
reboot
```

6.1.3 From SD Card

CHAPTER 7

Build your Image

CHAPTER 8

Getting Started: NetModule Linux

NetModule AG provides an open source Linux Distribution based on Yoctoproject's Reference distro "Poky".

If you are new into yoctoproject you might want to read this: Getting Started: The Yocto Project Overview

8.1 NetModule Addons

Board Support Packages

- Layer *meta-netmodule-bsp*: Recipes to provide hardware support

NetModule Distro

- Layer *meta-netmodule-distro*: NetModule HW specific package selection

Integration of OSTree

- Layer *meta-updater*

8.2 Image Types

netmodule-linux-release

- OSTree support
- clean build, ready to deploy

netmodule-linux-dev

- OSTree support
- Several tools a developer would find useful already integrated.

netmodule-linux-minimal

- Minimal Kernel and Image size

- RAM Disk Support

CHAPTER 9

OSTree

OSTree is a system for versioning updates of Linux-based operating systems. It can be considered as "git for operating system binaries".

OSTree Documentation

9.1 Description

9.1.1 Difference between a "normal" OS and an "atomic" OS

On a normal OS, the updates are handled by a package-manager. If an update is executed, the package-manager updates each package to the newest version available at the current time. This results in an unique set of packages after every update.

When you do an update on an atomic OS, you update the OS as a whole, giving you a specific set of packages every time.

9.1.2 Filesystem structure

An operating system deployed with ostree is always consistent with the "ostree commit". To make sure this is the case, the os has to be immutable. Ostree does this by creating a read-only bind-mount of the /usr folder.

OSTree uses UsrMov, this means that the folders "bin", "lib" and "sbin" are moved from the root to the /usr directory. On the root those folders are replaced with links to the new location under /usr. This is done, because it reduces the need of read-only bind-mounts to the single /usr directory.

The root of an ostree-deployment is located in: `/sysroot/ostree/deploy/<os-name>/deploy/<commit hash>`

9.1.3 System Partitioning

Because ostree mounts the /usr folder as read-only, we have two options to add additional software:

1. Create a new partition on the eMMC and mount it on lets say /data. | If done like this, new software can now be added to this data-folder.
2. Create an overlay over the /usr directory. If done like this, the /usr folder behaves like a normal read-write folder. This can be done with the command `ostree admin unlock --hotfix` **Note: The changes done like this are reverted after another ostree update. Note: This is generally not recommended for production systems.**

9.1.4 Boot Sequence

1. u-boot loads a uEnv.txt file which contains:

variable	contents
kernel_image	path to the kernel image
ramdisk_image	path to the ramdisk image
bootargs	the bootargs containing the path to the ostree-root.
kernel_image2	path to the fallback kernel image
ramdisk_image2	path to the fallback ramdisk image
bootargs2	the bootargs containing the path to the fallback ostree-root.

2. u-boot loads the kernel and ramdisk given by uEnv.txt
3. the ramdisk contains a script (/sbin/init) which prepares the rootfs.
4. the script runs `pivot_root` to switch from the ramdisk to the newly generated rootfs.
5. the script calls the /sbin/init of the new rootfs.

9.1.5 Update system with ostree via USB

1. Put the "*ostree_repo*" folder on a USB-pendrive (ext4 formated).
2. Plug the USB-pendrive into the nmhw21.
3. On the nmhw21 terminal type:

```
bash # full path of repository e.g. /mnt/ostree_repo
OSTREE_REPO_PATH= # name the repository e.g. update_repo or just repo
OSTREE_REPO_NAME=
```

```
ostree remote add OSTREE_REPO_NAME file://OSTREE_REPO_PATH --no-gpg-verify
ostree --repo=$OSTREE_REPO_PATH summary -u OSTREE_REFS_REPO=
ostree remote refs $OSTREE_REPO_NAME
ostree pull $OSTREE_REFS_REPO
ostree admin deploy $OSTREE_REFS_REPO
```

verify pending update

```
ostree admin status | grep pending
```

reboot to apply update

```
reboot
```

9.1.6 Update system with ostree via network

1. Connect to the network.

```
nmcli c mod ethernet ipv4.method auto
nmcli c up ethernet
```

2. Add the repositories.

```
ostree remote add nmrepo-stable https://nmrepo.netmodule.com/chbe/stable/ --no-gpg-
↪verify
ostree remote add nmrepo-unstable https://nmrepo.netmodule.com/chbe/unstable/ --no-
↪gpg-verify
```

3. Download the image.

There are multiple architectures and images available. The naming convention is:

`{YOCTO_VERSION}-{MACHINE}-{IMAGE_TYPE}` e.g. `dunfell-am335x-nmhw21-vcu`

Do only one of the following commands.

```
# Do this to get the newest stable image.
ostree pull nmrepo-stable dunfell-am335x-nmhw21-vcu
# Do this to get the newest unstable / nightly image.
ostree pull nmrepo-unstable dunfell-am335x-nmhw21-vcu
```

4. On the nmhw21 terminal type:

Do only one of the following commands.

```
# Do this to deploy the newest stable image.
ostree admin deploy nmrepo-stable:dunfell-am335x-nmhw21-vcu
# Do this to deploy the newest unstable / nightly image.
ostree admin deploy nmrepo-unstable:dunfell-am335x-nmhw21-vcu
```

CHAPTER 10

Artifacts for nmhw21

10.1 wic-file

Address in blocks of 512B	Content
0x0000	MBR (Partition Table)
0x0100 (128kB)	MLO
0x0300 (384kB)	u-boot.img
0x2000 (4096kB)	root-partition

CHAPTER 11

Artifacts overview

11.1 wic-file

The wic file is a flashable image file. When upgrading a system with a wic-file, the contents of the wic-file are mirrored to the mass storage device.

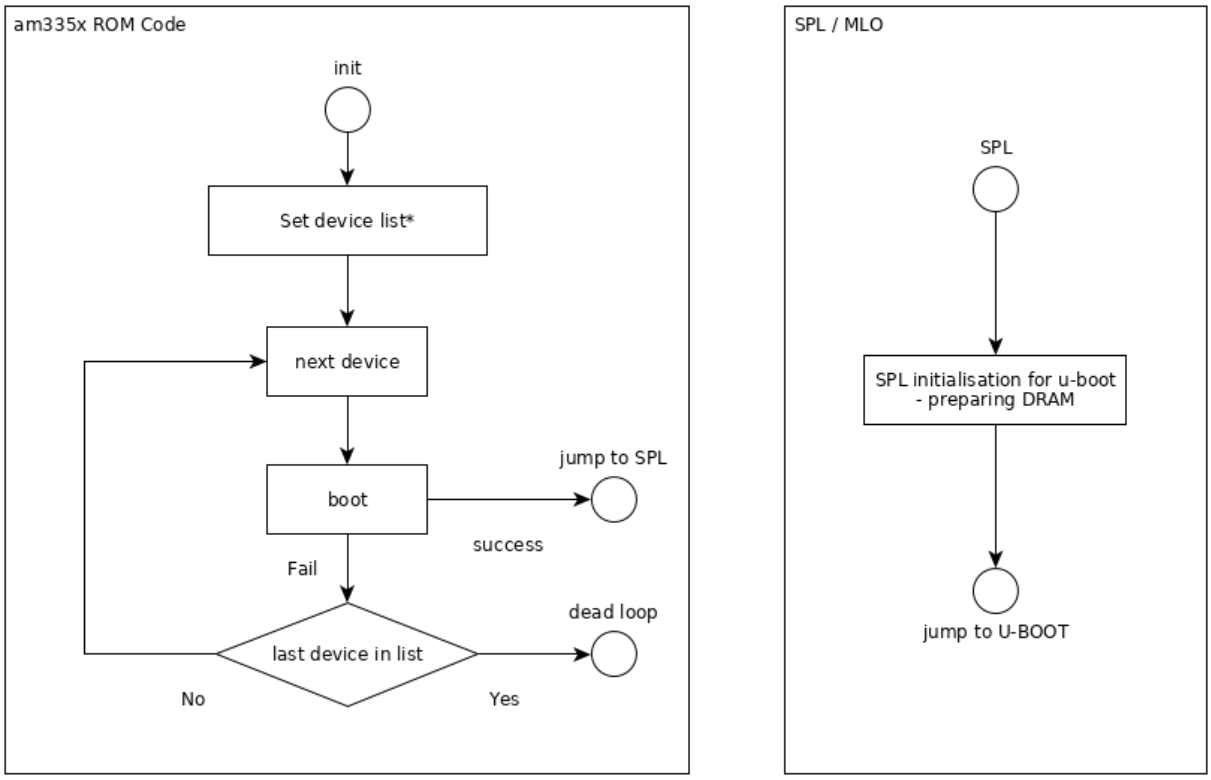
11.2 Kickstart/wks-file

The wks-file defines where the contents are placed in the wic-file. In our yocto build-system the file is located in *meta-netmodule-bsp/wic/*.wks*.

More Information: Yocto Project Reference Manual

CHAPTER 12

am335x Startup



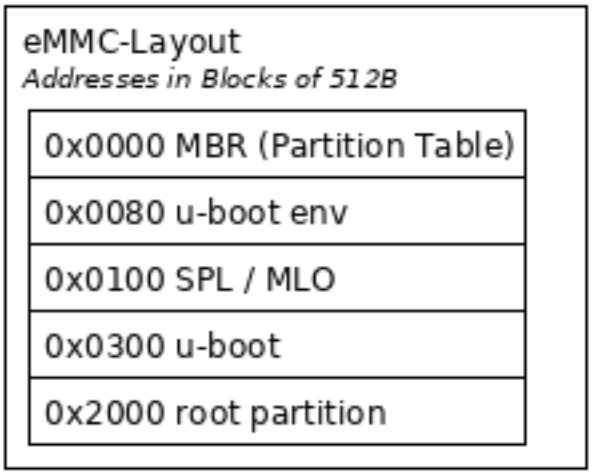
am335x-startup

HW	*list
nmhw21	MMC1, MMC0, UART0, USB0
nmhw21 (jumper on X200)	UART0, XIP, MMC0, NAND

See also: AM335x Technical Reference Manual -> Chapter 26 - Initialization (S. 5014)

CHAPTER 13

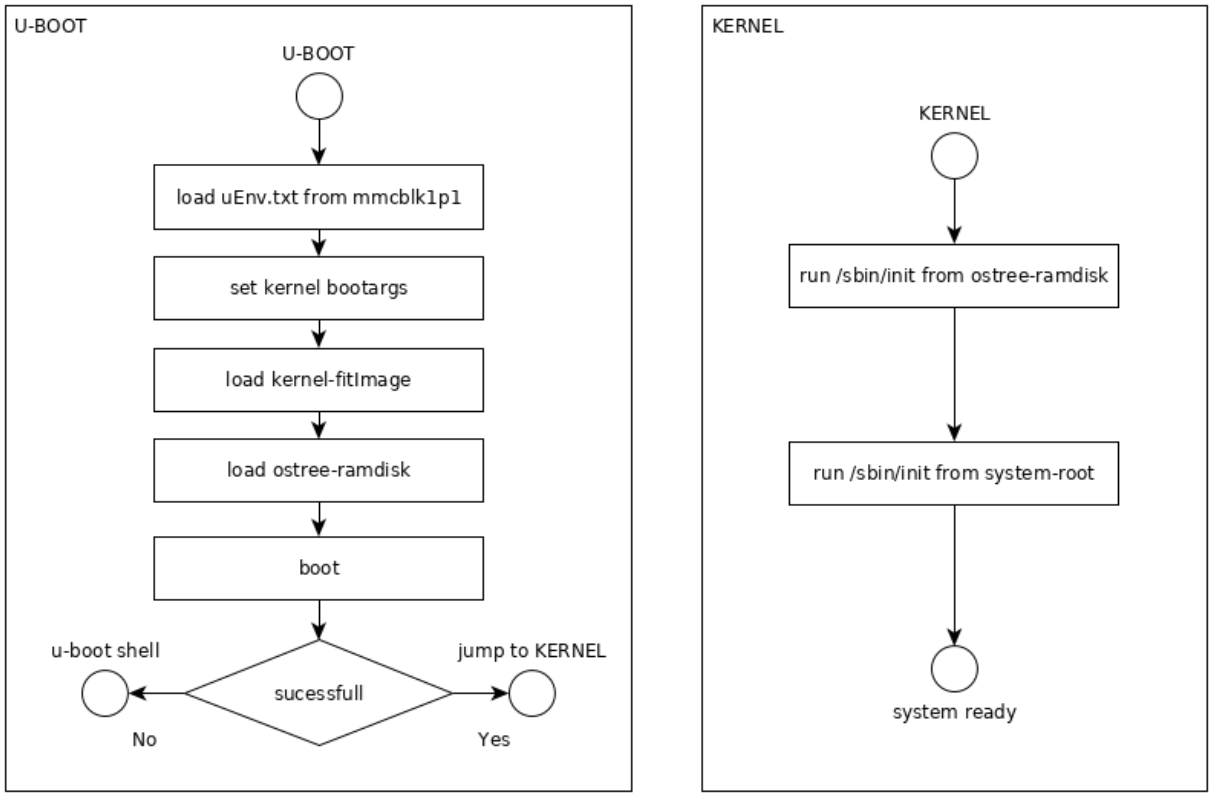
eMMC Contents



eMMC Contents

CHAPTER 14

Startup Sequence



Startup Graph

CHAPTER 15

Automatic Partitioning

When you flashed a ".wic"-file your system will only have one partition with almost no space. To make the filesystem usable, we provide an auto-partitioning tool called `nmhw-auto-part`.

15.1 Recommended settings

1. Run the script in interactive mode.

```
nmhw-auto-part -i
```

2. Press enter without entering values on all the steps.

15.2 Usage

```
nmhw-auto-part.sh [OPTION] 'devicename' 'size' 'mode' 'size'
```

15.2.1 Arguments

'devicename' device to use: e.g. /dev/sdb /dev/mmcblk1

'size' size of the first partition in MB (set 0 to keep current size)

'mode' 'data' or 'overlay'; type of partition to append

'size' size of the appended partition in MB (set to 0 to fill)

15.2.2 Options

```
-i interactive mode
-h help
```

You can either pass the four arguments 'devicename', 'size', 'mode' and 'size' or one option.

15.3 Examples

```
nmhw-auto-part /dev/mmcblk1 1024 overlay 0
```

The same as running the recommended settings.

15.4 How to undo the partitioning

1. Delete Partition information

```
rm -r /etc/nmhw-auto-part
reboot
parted /dev/mmcblk1 rm 2
```

4. Rerun the autopartition script.

Note: The first partition will keep its size.

15.4.1 Files in /etc

/etc/nmhw-auto-part/overlay This is an empty file. It indicates to the init script that it needs to mount an overlay.

/etc/nmhw-auto-part/data-partition This file contains the path of the device, which serves as the data partition. It indicates to the init script that it needs to mount the data partition.

CHAPTER 16

Bluetooth

For full documentation visit bluez.org.

16.1 Commands

16.2 Start and Discover nearby devices

```
root@am335x-vcu:~# bluetoothctl
[NEW] Controller 0C:B2:B7:11:61:9B am335x-vcu [default]
[NEW] Device 40:4E:36:55:DE:CE niconico

[bluetooth]# power on
Changing power on succeeded
[CHG] Controller 0C:B2:B7:11:61:9B Powered: yes

[bluetooth]# agent on
Agent registered

[bluetooth]# default-agent
Default agent request successful

[bluetooth]# discoverable on
[CHG] Controller 0C:B2:B7:11:61:9B Class: 0x200000
Changing discoverable on succeeded
[CHG] Controller 0C:B2:B7:11:61:9B Discoverable: yes

[bluetooth]# scan on
Discovery started
[CHG] Controller 0C:B2:B7:11:61:9B Discovering: yes
[NEW] Device 56:D2:37:FA:DC:8B 56-D2-37-FA-DC-8B
[NEW] Device 74:8D:3C:66:C9:7D 74-8D-3C-66-C9-7D
[NEW] Device 5A:D3:22:54:BD:6C 5A-D3-22-54-BD-6C
```

(continues on next page)

(continued from previous page)

```
[NEW] Device CC:5B:4F:F4:8A:2B fenix 3 HR
[CHG] Device 74:8D:3C:66:C9:7D RSSI: -80
[NEW] Device 6F:90:CD:32:DE:1B 6F-90-CD-32-DE-1B
[NEW] Device 46:FB:E1:BC:5F:C8 46-FB-E1-BC-5F-C8
[CHG] Device 6F:90:CD:32:DE:1B RSSI: -103
[CHG] Device 6F:90:CD:32:DE:1B RSSI: -85
[CHG] Device 5A:D3:22:54:BD:6C RSSI: -75
[CHG] Device 74:8D:3C:66:C9:7D RSSI: -82
[CHG] Device 40:4E:36:55:DE:CE RSSI: -52
[CHG] Device 5A:D3:22:54:BD:6C RSSI: -66

[bluetooth]# scan off
[CHG] Device E0:E5:CF:96:FA:09 RSSI is nil
[CHG] Device 40:4E:36:55:DE:CE RSSI is nil
[CHG] Device 46:FB:E1:BC:5F:C8 RSSI is nil

[CHG] Controller 0C:B2:B7:11:61:9B Discovering: no
Discovery stopped
```

CHAPTER 17

CAN

17.1 CAN Interfaces

Linux provides CAN driver for physical available CAN controller and for virtual created CAN adapter so called vcan.

17.1.1 Physical CAN Interface

Physical can interfaces depends on hardware and driver support. To check if physical can interfaces are available do:

```
ifconfig -a | grep can
```

Output should be similar to following:

```
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
can1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
```

17.1.2 Virtual CAN Interface - vcan

To bring up virtual can interface the kernel module vcan is required. Load vcan module:

```
modprobe vcan
```

And controls whether the module is loaded successfully:

```
lsmod | grep vcan
```

Output should be similar to following:

```
vcan                   16384  0
```

Now a virtual can interface vcan0 can be created:

```
ip link add dev vcan0 type vcan
ip link set vcan0 mtu 16
ip link set up vcan0
```

To bring up CAN FD interface mtu size must increased to 72:

```
ip link add dev vcan0 type vcan
ip link set vcan0 mtu 72
ip link set up vcan0
```

And again control new created virtual can interface:

```
ifconfig vcan0
```

Output should be similar to following:

```
vcan0     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

From this point the virtual can interface vcan0 can be used e.g. for SocketCAN.

17.2 SocketCAN

For linux based systems there are severals user space tools available to tunnel virtual can adapter over ethernet e.g. cannelloni or socketcand. Only cannelloni is treated here.

Bind virtual can adapter vcan0 to any counterpart:

```
cannelloni -I vcan0 -R <remote ip> -r <remote port> -l <local port>
```

For non blocking console append a `&` in the command above.

17.2.1 SocketCAN Between Two Linux Machines

Example use of SocketCAN between two machines.

Config	MACHINE 0	MACHINE 1
IP	192.168.1.100	192.168.1.200
Local cannelloni port	2000	2000

MACHINE 0

```
cannelloni -I vcan0 -R 192.168.1.200 -r 2000 -l 2000
```

MACHINE 1

```
cannelloni -I vcan0 -R 192.168.1.100 -r 2000 -l 2000
```

17.2.2 External Physical CAN Interface

Ensure that any can participant is on can bus. For communication verification a can PC interface is recommendend. Check also that physical bus is proper terminated with 120 Ohm impedance.

17.3 can-utils

can-utils provides severals tools to e.g. interact and monitor general can interfaces.

To send can frames to vcan0 command cansend can be used:

```
cansend <device> <can_frame>
# example
cansend vcan0 5A2#11.2233.445D556677.66
```

To dump can frames on a can interface use command candump:

```
candump <device>
# example
candump vcan0
```

CHAPTER 18

Chrony

Chrony is a versatile implementation of the Network Time Protocol (NTP). It can synchronise the system clock with NTP servers, reference clocks (e.g. GPS receiver), and manual input using wristwatch and keyboard. It can also operate as an NTPv4 (RFC 5905) server and peer to provide a time service to other computers in the network.

Chrony Website

Ensure that no other time daemon is running like ntp or ntimed to avoid conflicts.

18.1 Usage

Two programs are included in chrony, chronyd is a daemon that can be started at boot time and chronyc is a command-line interface program which can be used to monitor chronyd's performance and to change various operating parameters while it is running.

If chrony is already configured, time sources can be watched with:

```
chronyc sources
```

18.1.1 Chrony Configuration File

Chrony daemon configuration file is located in directory `/etc/chrony.conf`

18.1.2 NTP Servers

NTP time servers can be defined in configuration files. It is recommended to select time servers which are physicaly close to the device. A pool of time servers can be found at the NTP Pool Project.

Example configuration:

```
server 0.pool.ntp.org iburst
server 1.pool.ntp.org iburst
server 2.pool.ntp.org iburst
```

18.1.3 GNSS

Chrony is able to use GNSS signal and their delivered time. If GPSD is used as GNSS daemon, then chrony can access GPSD shared memory to get time data.

Example configuration with GPSD shared memory access:

```
refclock SHM 0 poll 1 refid GPS offset 0.0 delay 2 filter 16
```

18.1.4 RTC

Chrony can handle hardware real time clock (rtc), measure time drift and correct it automatically. Define rtc device in configuration file by adding device path.

```
rtcdevice /dev/rtc
```

18.2 More configuration option

Chrony is a powerfull tool and provides much more functionality than described here. For further information see Chrony Documentation

CHAPTER 19

Package Management using DNF

For full documentation visit dnf.readthedocs.io <https://dnf.readthedocs.io/en/latest/index.html>‘_.

19.1 Setup

Add the netmodule package repository

```
mkdir /etc/yum.repos.d
touch /etc/yum.repos.d/oe-packages.repo
```

- Add following lines to `/etc/yum.repos.d/oe-packages.repo`

```
[oe-packages]
baseurl=https://nmrepo.netmodule.com/chbe/rpm/
gpgcheck=False
```

19.2 Commands

- `dnf search <pkg-name>` - Search package
- `dnf install <pkg-name>` - Install package
- `dnf remove <pkg-name>` - Remove package

19.3 OSTree compatibility

To use dnf with ostree you have to do the following steps:

1. mount an overlay-fs You might use the auto-partition-script with the recommended settings.
2. Add symbolic link to /lib/rpm

```
ln -s /usr/lib/rpm /var/lib/rpm
```

Hint: If this fails, remove the `/var/lib/rpm` directory and rerun this step.

CHAPTER 20

Ethernet

20.1 Overview

NetModule OEM Linux Distribution comes with built-in support for a number of Ethernet devices. This includes support for hardware solutions as simple as single port standard Ethernet PHY and very complex hardware architecture involving cascading Ethernet switches with external PHYs.

20.2 Supported scenarios

20.2.1 Basic setup

Single network for internal (umnet0) end external (lan0, broadr0 and broadr1) interfaces. Main IP address assigned to eth0. Common broadcast domain.

20.2.2 Multiple isolated interfaces

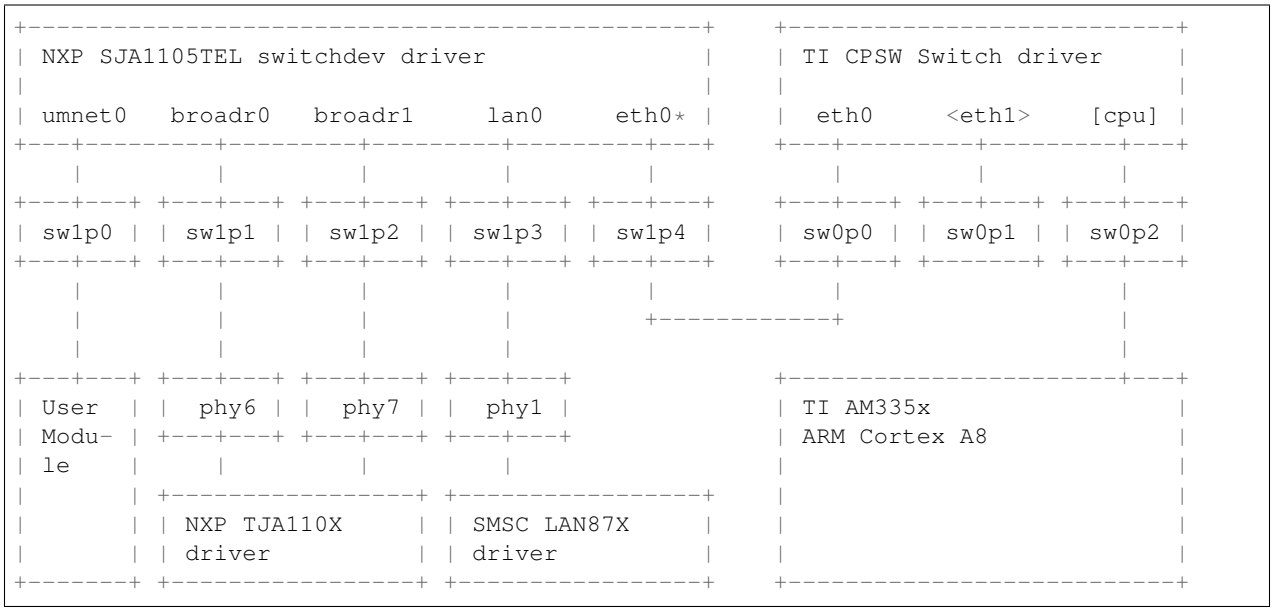
Up to 4 isolated network connections thanks to SJA1105TEL VLAN tagging. IP addresses assigned to VLAN aware interfaces on top of eth0. See *example*.

20.3 Supported hardware

20.3.1 NMHW21

- TI Common Platform Ethernet Switch (CPSW)
- NXP SJA1105TEL Five- Ports AVB & TSN Automotive Ethernet Switch
- NXP TJA1100 100BASE-T1 PHY for Automotive Ethernet

- SMSC LAN8710/LAN8720 PHY



20.4 NXP SJA1105TEL switchdev driver

The SJA1105TEL switchdev driver functionality: - Ports are available in user space as netdevs (e.g. broadr1, lan0) - Hardware statistics and standard information about devices (ethtool) - Rx/Tx and error statistics (ifconfig) - Hardware offloading (bridge fdb) - VLAN tagging/bridging support (brctl, bridge vlan)

The SJA1105TEL switchdev driver limitations: - No VLAN double tagging

20.4.1 Hardware offloading

The idea is to offload the L2 data forwarding (switching) path from the kernel to the switch device by mirroring bridge FDB entries down to the device. An FDB entry is the {port, MAC, VLAN} tuple forwarding destination.

Static bridge FDB entries are installed, for example, using bridge command:

```
$ bridge fdb add ADDR dev DEV [vlan VID] [self]
```

The SJA1105TEL switch device has 1024 entries in the L2 address lookup table. Parts of the table can be statically configured, e.g.:

```
$ bridge fdb add 00:1b:21:2f:fa:1f dev lan0
```

Some of the entries are dynamically learned during operation. Show all valid FDB entries:

```
$ bridge fdb show
```

Loaded entries never time-out and cannot be replaced by learned entries. They have to be removed manually, e.g.:

```
$ bridge fdb del 00:1b:21:2f:fa:1f dev lan0
```

20.4.2 VLAN tagging/bridging

Each port of the SJA1105TEL switch has assigned a PVID in MAC Configuration table. The respective entry in the VLAN Lookup table must be loaded and the flag of the port set to indicate VLAN membership.

A Port VLAN ID (PVID) is a default VLAN ID used to tag untagged incoming frames. In addition to the PVID mandatory definition, a port can be assigned to other VLANs. In this case tagged incoming frames are expected.

E.g. isolate ports 1 (broadr0) and 2 (broadr1) for external traffic by creating a bridge with PVID 10:

```
$ brctl addbr bridge0
$ ip link set dev bridge0 type bridge vlan_default_pvid 10
$ brctl addif bridge0 broadr0
$ brctl addif bridge0 broadr1
$ bridge vlan
port    vlan ids
bridge0  10 PVID Egress Untagged

broadr0  10 PVID Egress Untagged

broadr1  10 PVID Egress Untagged
```

E.g. enable VLAN filtering on bridge0 (untagged incoming frames are dropped):

```
$ ip link set dev bridge0 type bridge vlan_filtering 1
```

E.g. add VLAN for tagged incoming frames and tagged egress:

```
$ bridge vlan add vid 12 tagged dev broadr0
$ bridge vlan add vid 12 tagged dev broadr1
$ bridge vlan
port    vlan ids
bridge0  10 PVID Egress Untagged
         12

broadr0  10 PVID Egress Untagged
         12

broadr1  10 PVID Egress Untagged
```

20.5 NXP TJA110X PHY driver

The driver exposes configuration options through sysfs.

Checking link status of a TJA1100 PHY.

E.g. broadr0, cable connected:

```
# cat /sys/bus/mdio_bus/devices/4a101000.mdio\:02/configuration/link_status
up
```

E.g. broadr1, cable disconnected:

```
# cat /sys/bus/mdio_bus/devices/4a101000.mdio\:03/configuration/link_status
down
```

Both TJA110X PHYs are configured as slaves by default.

E.g. check master/slave configuration of broadr0, default configuration:

```
# cat /sys/bus/mdio_bus/devices/4a101000.mdio\:02/configuration/link_status
slave
```

E.g. set broadr1 as master (a counterpart has to be set as slave):

```
# echo master > /sys/bus/mdio_bus/devices/4a101000.mdio\:03/configuration/master_cfg
# cat /sys/bus/mdio_bus/devices/4a101000.mdio\:03/configuration/link_status
master
```

The TJA1100 PHY driver can give SNR class of a connection.

E.g. check connection quality of broadr0 to a counterpart over short good quality cable:

```
# cat /sys/bus/mdio_bus/devices/4a101000.mdio\:02/configuration/snr_class
Class G SQI (very good link)
```

E.g. check connection quality of broadr1 to a counterpart over long poor quality cable:

```
# cat /sys/bus/mdio_bus/devices/4a101000.mdio\:03/configuration/snr_class
Class B SQI (unstable link)
```

20.6 User space tools and configuration

Check available interfaces:

```
$ ip link show
```

List IP addresses:

```
$ ip address show
```

List routes:

```
$ ip route show
```

Deactivate a link layer device:

```
$ ip link set dev devicename down
```

Activate a device:

```
$ ip link set dev devicename up
```

Print current settings of the specified device

```
$ ethtool devname
```

Print statistics of the specified device

```
$ ethtool -S devname
```

20.7 Multiple isolated interfaces example

Here are the steps necessary to create 3 IP interfaces able to connect to 3 isolated networks:

- Internal, 192.168.1.0/24 - connection between User Module and Linux
- Ethernet LAN, 172.16.71.0/24 - standard Ethernet connection to Linux
- BroadR LAN, 172.17.1.0/24 - BroadR connection to Linux

1. Set 0.0.0.0 ipv4 address for eth0

```
# nmcli -p con show ethernet
...
ipv4.addresses:                         0.0.0.0/32
...
```

2. Add vlan aware interfaces on top of eth0 using nmcli (routing added automatically)

```
# nmcli con add type vlan con-name eth0.16 dev eth0 id 16 ip4 172.16.71.1/24
# nmcli con add type vlan con-name eth0.17 dev eth0 id 17 ip4 172.17.1.10/24
# nmcli con add type vlan con-name eth0.19 dev eth0 id 19 ip4 192.128.1.1/24
```

3. Create bridge with eth0* as trunk port and separated access ports for all networks

```
# brctl addbr trunk0
# brctl addif trunk0 umnet0
# brctl addif trunk0 broadr0
# brctl addif trunk0 broadr1
# brctl addif trunk0 lan0
# brctl addif trunk0 eth0*
```

4. Update VLAN IDs

```
# bridge vlan add dev eth0* vid 16 tagged
# bridge vlan add dev lan0 vid 16 pvid untagged

# bridge vlan add dev eth0* vid 17 tagged
# bridge vlan add dev broadr0 vid 17 pvid untagged
# bridge vlan add dev broadr1 vid 17 pvid untagged

# bridge vlan add dev eth0* vid 19 tagged
# bridge vlan add dev umnet0 vid 19 pvid untagged
```

5. Clean up default PVID 1

```
# bridge vlan del dev umnet0 vid 1
# bridge vlan del dev broadr0 vid 1
# bridge vlan del dev broadr1 vid 1
# bridge vlan del dev lan0 vid 1
```

6. Verify

```
# bridge vlan
port    vlan ids
umnet0   19 PVID Egress Untagged

broadr0  17 PVID Egress Untagged
```

(continues on next page)

(continued from previous page)

```
broadr1  17 PVID Egress Untagged

lan0     16 PVID Egress Untagged

eth0*    1 PVID Egress Untagged
         16
         17
         19

trunk0   1 PVID Egress Untagged
```

7. Analyse traces from eth0 (all traffic should be tagged with VLAN ID assigned to a port)

```
# tcpdump -nnei eth0 -vvv
```

20.8 References

- kernel.org
- dsa
- phy
- netdevices
- switchdev

CHAPTER 21

Updating firmware

The NetModule Linux system provides a tool to upgrade the firmware of the different modules in the system.

The firmware packages are not available in the reference image but the latest supported firmware versions are part of each software release.

21.1 Supported modules

21.1.1 WWAN modules

- u-blox TOBY-L210

21.1.2 GNSS modules

- u-blox NEO-M8L

21.2 Firmware Location

The firmwares officialy supported by NetModule are available at https://nmrepo.netmodule.com/chbe/fwupdate/

21.3 Preparation

Before upgrading a module, it is recommended to check if it has been properly configured and started

This can can be done using the following commands:

For GNSS modules:

```
systemctl status gnss-config
```

For WWAN modules:

```
systemctl status wwan-config@wwan0
```

When the modules are properly configured, the output shows one of the two following lines:

```
Active: active (running)
Active: active (exited)
```

Once the device is started, the firmware version can be read with the following commands:

For GNSS modules:

```
cat /run/gnss/gnss0.config
```

For WWAN modules:

```
cat /run/wwan/wwan0.config
```

21.4 Usage

The update tool can fetch the new firmware from the local device or via http from remote firmware.

21.4.1 Local firmware

To upgrade using a local firmware, the firmware must be copied to the target first. The following command can then be executed:

```
nmhw-fwupdate <device> <firmware.tar.gz>
```

Where <device> is the device name as reported by *list-devices* and <firmware> is the relative or full path to the firmware archive.

Examples :

```
# Updating gnss firmware
nmhw-fwupdate gnss0 UBX_M8_301_ADR_421_NEO_M8L.tar.gz

# Updating wwan firmware
nmhw-fwupdate wwan0 TOBY-L210-03S-01_FW16.19_A01.02_IP.tar.gz
```

21.4.2 Remote firmware (HTTP)

If the firmware is available on an HTTP server (reachable from the target), the following command can be used:

```
nmhw-fwupdate <device> <http link>
```

Where <device> is the device name as reported by *list-devices* and <http link> is the link to the firmware on the HTTP server.

Examples :

```
# Updating gnss firmware
nmhw-fwupdate gnss0 https://nmrepo.netmodule.com/chbe/fwupdate/UBX_M8_301_ADR_421_NEO_
↪M8L.tar.gz

# Updating wwan firmware
nmhw-fwupdate wwan0 https://nmrepo.netmodule.com/chbe/fwupdate/TOBY-L210-03S-01_FW16.
↪19_A01.02_IP.tar.gz
```

CHAPTER 22

GNSS

The Global Navigation Satellite System can be accessed using different tools: gpsd provides a useful toolset and the gnss-mgr provides configuration possibilities.

22.1 gpsd

gpsd is used as the interface between the GNSS receiver and other location aware applications. Multiple applications can access the GNSS receiver via TCP connections on port 2947 at the same time, solving the problem of multiple applications requiring access to the same tty interface. gpsd includes several tools to interface to the GNSS receiver, like cpgs, gpsctl, gpscat and ubxtool.

22.2 NEO-M8L

The NEO-M8L module is a GNSS receiver by u-blox with the following key features:

- Automotive Grade
- GPS / QZSS, GLONASS, Galileo and BeiDou
- Dead Reckoning using built in IMU

22.3 gnss-mgr

`gnss-mgr` provides GNSS receiver configuration possibilities and replaces the obsolete `gnss-config`, `gnss-save-on-shutdown` and `neom8tool`. This tool operates directly on the serial interface *before* gpsd is staring up and comes with different functionalities like:

- **initialization (runs on each boot-up)**
  - configuring the communciation bitrate of the receiver if necessary

  - configuring the used NMEA protocol version if necessary
  - clearing latest receiver state (save on shutdown)
- save on shutdown (SoS) features like persisting/clearing/verifying receiver state to/from internal storage
- configuring the receiver using a configuration file (in ini file format, see below)
- **control functions**
  - persist the actual configuration into non-volatile storage
  - cold-start the receiver
  - resetting the receiver's configuration to default

The gnss-mgr is set up as systemd service (`gnss-mgr.service`).

On each boot-up the `gnss-mgr` checks and if necessary configures the bitrate and the NMEA protocol version. The `gnss-mgr` configures the recevier with volatile parameters, i.e. the configuration is not persisted and when the power is cut, the receiver looses your configuration (the receiver then re-applies its internally stored setup at the next power-up). Additionally after successfully starting up the gnss services, some basic information about the GNSS receiver can be found at `/run/gnss/gnss0.conf`.

```
root@am335x-nmhw21:~# cat /run/gnss/gnss0.config
Vendor:                             ublox
Model:                              NEO-M8L-0
Firmware:                           ADR 4.21 (Deprecated)
ubx-Protocol:                       19.20
Supported Satellite Systems:        GPS;GLO;GAL;BDS
Supported Augmentation Services:    SBAS;IMES;QZSS
SW Version:                         EXT CORE 3.01 (1ec93f)
HW Version:                         00080000
```

22.3.1 Capabilities

The command `gnss-mgr --help` shows the entire capabilities of the `gnss-mgr`:

```
root@am335x-nmhw21:~# gnss-mgr --help
usage: gnss-mgr [-h] [-V] [-v] [-q] device {init,sos,config,control} ...

Manages GNSS modem

positional arguments:
  device                local serial device to which GNSS modem is connected
                        (e.g. /dev/gnss0)

optional arguments:
  -h, --help            show this help message and exit
  -V, --version         show program's version number and exit
  -v, --verbose         be verbose, show debug output
  -q, --quiet           be quiet, only show warnings and errors

command:
  {init,sos,config,control}
                        select command
    init                sets up GNSS modem
    sos                 save on shutdown operations
    config              configures GNSS modem
    control             performs GNSS modem control function
```

Each command (init, sos, config and control) provides a sub-help:

```
root@am335x-nmhw21:~# gnss-mgr /dev/gnss0 init --help
usage: gnss-mgr device init [-h] [-f RUNFILE]

optional arguments:
  -h, --help            show this help message and exit
  -f RUNFILE, --file RUNFILE
                        path to run file

----------------------------

root@am335x-nmhw21:~# gnss-mgr /dev/gnss0 sos --help
usage: gnss-mgr device sos [-h] {save,clear}

positional arguments:
  {save,clear}  selects sos operation to perform

optional arguments:
  -h, --help    show this help message and exit

----------------------------

root@am335x-nmhw21:~# gnss-mgr /dev/gnss0 config --help
usage: gnss-mgr device config [-h] [-f CONFIGFILE]

optional arguments:
  -h, --help            show this help message and exit
  -f CONFIGFILE, --file CONFIGFILE
                        path to config file

----------------------------

root@am335x-nmhw21:~# gnss-mgr /dev/gnss0 control --help
usage: gnss-mgr device control [-h] {cold-start,persist,factory-reset}

positional arguments:
  {cold-start,persist,factory-reset}
                        selects action to perform

optional arguments:
  -h, --help            show this help message and exit
```

22.4 Configuring the GNSS Receiver

The `gnss-mgr` configures the GNSS receiver by using configuration files in ini format. Writing the configuration is managed by feeding the appropriate file to the gnss-mgr.

> **Warning:** A misconfigured GNSS with Dead Reckoning (DR) generates worse results as a GNSS receiver only!

Note: When running the GNSS receiver at 9600 baud some UBX packets might be lost which results in overall worse GNSS performance. Therefore it is strongly recommended to let the receiver operate with 115200 baudto.

Additionally it is recommended to configure the GNSS receiver persistent (see example *Persistent configuration of the GNSS receiver* below.

22.4.1 Configuration files

Use the configuration file `/etc/gnss/gnss0.conf` as template to properly configure the GNSS receiver.

The following example of gnss0 config file shows the configuration capabilities of the `gnss-mgr`. The comments show how to setup each parameter:

```
root@am335x-nmhw21:~# cat /etc/gnss/gnss0.conf
#
# This file is part of gnss-mgr service
# To make changes, edit the values in this file and reload
# gnss-mgr service.
#
# Any empty value will use the default or persistent configuration
# of the receiver.

[default]
# Indicates the version of this config file, it should not be modified.
# If unsure of its value, sample config file can always be found in
# /usr/etc/gnss/
#
# Change history
# version 3: Added field navigation.dead-reckoning
version=3

# Select measurement and navigation output rate
# Allowed values : 1, 2  [Hz]
update-rate=
#update-rate=1


#
# Navigation settings
#
[navigation]
# Enables or disables dead reckoning
# Supported values
#   enable, disable
dead-reckoning=
#dead-reckoning=disable

# Selects dynamic mode
# Supported values:
#   stationary, vehicle
mode=
#mode=vehicle

#
# Selects GNSS systems
# Allowed values:
#   GPS;GLONASS;SBAS
#   GPS;Galileo;Beidou;SBAS
systems=
```

(continues on next page)

(continued from previous page)

```
#systems=GPS;GLONASS;SBAS
#systems=GPS;Galileo;Beidou;SBAS


#
# Installation settings
# For details on this section, see the relevant documentation
#
[installation]

#
# IMU orientation in degrees [°]
#   yaw: value in degrees (0 to  360)
#   pitch: value in degrees (-90 to  90)
#   roll: value in degrees (-180 to 180)
yaw=
pitch=
roll=

# Lever arm lengths in meters [m]
# Format x;y;z
# Example:
#   vrp2antenna=1.0;1.5;0.3
vrp2antenna=
vrp2imu=
```

22.4.2 Lever Arm Lengths

The following figure shows an installation example and how to configure the receiver's lever arm lengths.

Fig. 1: Installation and positioning

LeverConfiguration with values

- vrp2antenna=1.6;0;1.7
- vrp2imu=2.1;0.6;0.8

22.4.3 Examples

Configuring the GNSS receiver:

```
gnss-mgr /dev/gnss0 config -f /etc/gnss/gnss0.conf
```

Persisting the configuration of the GNSS receiver:

```
gnss-mgr /dev/gnss0 control persist
```

Resetting the receiver's configuration to default:

```
gnss-mgr /dev/gnss0 control factory-reset
```

22.4.4 Troubleshooting

If the receiver is no longer accessible, the issue most seen is that baud rates of serial interface (ttyS3) and receiver are not matching.

22.5 Save on Shutdown

The NEO-M8L GNSS receiver can be instructed to save its current state to the internal non-volatile memory and restore it after a power cycle.

Note that the receiver state and the receiver configuration must be distinguished: Saving the state before the system is shut-down can help your GNSS receiver to get a faster fix after booting. Save on shutdown does however not save any configuration done by the user. The receiver configuration needs to be saved manually to the receiver's internal non-volatile memory (only once), see Example *Persiting the configuration of the GNSS receiver*.

The `gnss-mgr` service instructs the GNSS receiver to save its state whenever your linux system receives a shutdown or reboot instruction. Upon reboot the same service logs if the GNSS receiver state has been successfully restored.

22.5.1 Examples

Storing the state to the GNSS receiver's internal storage:

```
gnss-mgr /dev/gnss0 sos save
```

Clearing the state in the GNSS receiver's internal storage:

```
gnss-mgr /dev/gnss0 sos clear
```

22.6 Firmware update (specific to u-blox NEO-M8L)

The firmware of the module u-blox NEO-M8L can be upgraded as explained in *Updating firmware*.:

22.7 Testing

To test the GNSS function connect an active GNSS antenna to X3300 "GNSS".

Run "cgps" tool

Your output should look like this. Typically it takes 3..20 seconds for a fix.

```
┌──────────────────────────────────────────────────────────────────────────────┐
│    Time:       2018-07-05T06:49:00.000Z  │ PRN:   Elev:  Azim:  SNR:  Used: │
│    Latitude:    47.31890666 N            │   6     18     082    28      Y  │
│    Longitude:    7.97375949 E            │  17     15     039    30      Y  │
│    Altitude:   1641.076 ft               │  19     35     052    27      Y  │
│    Speed:      0.14 mph                  │  32     32     305    16      Y  │
│    Heading:    0.0 deg (true)            │  66     27     311    23      Y  │
│    Climb:      0.00 ft/min               │  74     43     076    28      Y  │
│    Status:     3D FIX (3 secs)           │  84     46     063    21      Y  │
│    Longitude Err:   +/- 33 ft            │                                  │
└──────────────────────────────────────────────────────────────────────────────┘
```

(continues on next page)

(continued from previous page)

```
    Latitude Err:    +/- 111 ft
    Altitude Err:    +/- 200 ft
    Course Err:      n/a
    Speed Err:       +/- 152 mph
    Time offset:     7176.328
    Grid Square:     JN37xh
```

22.8 Access gps interface

If direct interface access is required, the gps device is available under `/dev/gnss0`. Be aware that `gnss0` is just a symlink to the real device managed by udev. If you need to know the real device, follow the symlink with:

Example output:

```
root@am335x-nmhw21:~# ls -l /dev/gnss0
lrwxrwxrwx 1 root root 5 Jul  2 07:06 /dev/gnss0 -> ttyS3
```

CHAPTER 23

GSM

See *WWAN*.

CHAPTER 24

IMU

ST provides LSM6DSx driver which provides two different ways of reading out IMU data.

24.1 Polling mode

Polling mode is the simplest IMU driver configuration, where data is read out on request. This mode does not support hardware timestamping.

24.1.1 Accelerometer

Get raw value of z-axis

```
$ cat /sys/bus/iio/devices/iio\:device0/in_accel_z_raw
```

Get y-axis scale value

```
$ cat /sys/bus/iio/devices/iio\:device0/in_accel_y_scale
```

Multiply the two to get the acceleration in m/s2. For z-axis this should be around 9.81m/s2.

Example:

```
$ cat /sys/bus/iio/devices/iio\:device0/in_accel_z_raw
16499
$ cat /sys/bus/iio/devices/iio\:device0/in_accel_y_scale
0.000598

# --> 9.8664
```

24.1.2 Gyro

Get raw value of x-axis

```
$ cat /sys/bus/iio/devices/iio\:device1/in_anglvel_x_raw
```

24.2 Buffered mode

Buffered mode enables full driver functionality, but requires hardware interrupts to be processed by the driver, which consumes more CPU load. This mode supports hardware timestamping.

24.2.1 Configuration

Buffered mode requires configuration. Following is a typical configuration to receive all the data IMU provides, including hardware timestamps:

Accelerometer

```
# Disable buffered mode

$ echo 0 > /sys/bus/iio/devices/iio\:device0/buffer/enable

# Subscribe to data elements

$ echo 1 > /sys/bus/iio/devices/iio\:device0/scan_elements/in_accel_x_en
$ echo 1 > /sys/bus/iio/devices/iio\:device0/scan_elements/in_accel_y_en
$ echo 1 > /sys/bus/iio/devices/iio\:device0/scan_elements/in_accel_z_en
$ echo 1 > /sys/bus/iio/devices/iio\:device0/scan_elements/in_timestamp_en

# Set sampling frequency

$ echo 13 > /sys/bus/iio/devices/iio\:device0/sampling_frequency

# Set buffer length

$ echo 2 > /sys/bus/iio/devices/iio\:device0/buffer/length

# Enable buffered mode

$ echo 1 > /sys/bus/iio/devices/iio\:device0/buffer/enable
```

Gyro

```
# Disable buffered mode

$ echo 0 > /sys/bus/iio/devices/iio\:device1/buffer/enable

# Subscribe to data elements

$ echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_anglvel_x_en
```

(continues on next page)

(continued from previous page)

```
$ echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_anglvel_y_en
$ echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_anglvel_z_en
$ echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_timestamp_en

# Set sampling frequency

$ echo 13 > /sys/bus/iio/devices/iio\:device1/sampling_frequency

# Set buffer length

$ echo 2 > /sys/bus/iio/devices/iio\:device1/buffer/length

# Enable buffered mode

$ echo 1 > /sys/bus/iio/devices/iio\:device1/buffer/enable
```

24.2.2 Data

Reading out IMU data in buffered mode is done in a different way comparing to polling mode. In polling mode data is provided via driver SysFS nodes, while in buffered mode data is read out via dedicated char devices.

Accelerometer

```
$ hexdump /dev/iio\:device0
```

Gyro

```
$ hexdump /dev/iio\:device1
```

24.2.3 Reconfiguration

Buffered mode does not allow reconfiguration on the fly. However, it is allowed to temporarily disable buffered mode while IMU char device is opened, which gives a possibility to reconfigure IMU driver. For example, if some application is used to read the data from accelerometer (see example above), it is possible to change its sampling frequency without closing the application:

```
$ echo 0 > /sys/bus/iio/devices/iio:device0/buffer/enable
$ echo 13 > /sys/bus/iio/devices/iio:device0/sampling_frequency
$ echo 1 > /sys/bus/iio/devices/iio:device0/buffer/enable
```

CHAPTER 25

LEDs

NMHW21 provides four LEDs that can be controlled by user application or via command line.

The LEDs are described as follows:

- Indicator onboard (ind)
- Status onboard (status)
- Indicator user-interface (ind ui)
- Status user-interface (status ui)

25.1 Standard LED behavior

On startup (during bootloader/system-boot) the LEDs follow a specific pattern. During the startup the LEDs on the user-interface and the onboard LEDs are synced.

‘ind’ = ‘ind ui’
‘status’ = ‘status ui’

- start of the bootloader (hard coded into u-boot)

ind - red
status - red

- start of kernel (defined in the device tree)

ind - off
status - orange

25.1.1 Hard reset

On a hard reset the onboard status LED starts orange.

25.2 LED: PCB Ind

Green

```
$ echo 1 > /sys/class/leds/ind\:green/brightness
$ echo 0 > /sys/class/leds/ind\:red/brightness
```

Orange

```
$ echo 1 > /sys/class/leds/ind\:green/brightness
$ echo 1 > /sys/class/leds/ind\:red/brightness
```

Red

```
$ echo 0 > /sys/class/leds/ind\:green/brightness
$ echo 1 > /sys/class/leds/ind\:red/brightness
```

Off

```
$ echo 0 > /sys/class/leds/ind\:green/brightness
$ echo 0 > /sys/class/leds/ind\:red/brightness
```

25.3 LED: PCB Status

Green

```
$ echo 1 > /sys/class/leds/status\:green/brightness
$ echo 0 > /sys/class/leds/status\:red/brightness
```

Orange

```
$ echo 1 > /sys/class/leds/status\:green/brightness
$ echo 1 > /sys/class/leds/status\:red/brightness
```

Red

```
$ echo 0 > /sys/class/leds/status\:green/brightness
$ echo 1 > /sys/class/leds/status\:red/brightness
```

Off

```
$ echo 0 > /sys/class/leds/status\:green/brightness
$ echo 0 > /sys/class/leds/status\:red/brightness
```

25.4 LED: UI Ind

Green

```
$ echo 1 > /sys/class/leds/ui\:ind\:green/brightness
$ echo 0 > /sys/class/leds/ui\:ind\:red/brightness
```

Orange

```
$ echo 1 > /sys/class/leds/ui\:ind\:green/brightness
$ echo 1 > /sys/class/leds/ui\:ind\:red/brightness
```

Red

```
# echo 0 > /sys/class/leds/ui\:ind\:green/brightness
$ echo 1 > /sys/class/leds/ui\:ind\:red/brightness
```

Off

```
# echo 0 > /sys/class/leds/ui\:ind\:green/brightness
$ echo 0 > /sys/class/leds/ui\:ind\:red/brightness
```

25.5 LED: UI Status

Green

```
$ echo 1 > /sys/class/leds/ui\:status\:green/brightness
$ echo 0 > /sys/class/leds/ui\:status\:red/brightness
```

Orange

```
$ echo 1 > /sys/class/leds/ui\:status\:green/brightness
$ echo 1 > /sys/class/leds/ui\:status\:red/brightness
```

Red

```
$ echo 0 > /sys/class/leds/ui\:status\:green/brightness
$ echo 1 > /sys/class/leds/ui\:status\:red/brightness
```

Off

```
$ echo 0 > /sys/class/leds/ui\:status\:green/brightness
$ echo 0 > /sys/class/leds/ui\:status\:red/brightness
```

CHAPTER 26

Linux System Logging

Our distribution supports user space and kernel log messages and uses journald as system logger. Its configuration file provides a lot of parameters and can be found under `/etc/systemd/journald.conf` (for the full detailed information, please check [journald.conf(5)](https://www.man7.org/linux/man-pages/man5/journald.conf.5.html)). This section describes the most important parameters and information regarding logging.

26.1 Accessing Log Files

To read out journals the command line tool journalctrl is provided: `journalctl`

To follow latest log messages pass argument -f: `journalctl -f`

Show filtered by units: `journalctl -u <your-unit.service>`

More information in journalctrl man page

26.2 Storage of the Logs

There are two modes about how logs can be stored: - volatile = logs are stored to RAM - persistent = logs are stored to flash

NOTE: The flash memory has limited write cycles which needs to be taken into account for the devices lifetime.

IMPORTANT: To not stress unnecessary the flash memory, our distribution logs as default the messages into RAM = volatile.

The logging system can be tailored to the need of the application. Full system logging makes analyzing an issue simpler and purposeful but it may cost valuable disk space and perfomance as well.

The follow subsection explains the most important configuration settings.

26.3 Configuration Parameters

In the config file `/etc/systemd/journald.conf` all parameters can be changed and tailored to the application's need. Nevertheless, the following parameters are the most important to understand:

- **Storage=** Can be either "volatile", "persistent", "auto" or "none". - volatile (*default*)

  The logs will be stored in memory below */run/log/journal*. After a reboot the logs will be gone. *The storage parameters with the "Runtime" prefix apply here.*

  - **persistent** The logs will be stored on the disk below */var/log/journal*. After a reboot the logs will still be there. *The storage parameters with the "System" prefix apply here.*
  - **auto** If the folder `/var/log/journal` exists, the behavior is the same as with "persistent" otherwise it behaves like "volatile"
  - **none** No logs will be saved. Only forwarding will be done (if enabled).

- **SystemMaxUse= / RuntimeMaxUse=** Control how much disk / memory space the journal may use up at most. Specify values in bytes or use K, M, G, T, P, E as units. *Default values: SystemMaxUse=64M, RuntimeMaxUse=16M*
- **SystemKeepFree= / RuntimeKeepFree=** Control how much disk / memory space the journal may keeps free. Specify values in bytes or use K, M, G, T, P, E as units. Systemd-journald will respect both limits (KeepFree/MaxUse) and use the smaller of the two values. *Default values: SystemKeepFree=350M, RuntimeKeepFree=not set*
- **SyncIntervalSec=** The timeout before synchronizing journal files to disk. This setting takes time values which may be suffixed with the units "m" for minutes. *Default value = 5min*
- **ForwardToSyslog=** Use "yes" or "no". Enables forwarding to the old syslog (/var/log/messages). If this is enabled, there will still be messages written to the disk regardless of the "Storage=" parameter. *Default = no*
- **ForwardToConsole=** Use "yes" or "no". Useful for debugging purposes *Default = no*

26.3.1 How to change the settings

If you want to change the default settings like for example persisting the log files, the following steps need to be done...

1. Change the storage parameter in the config file `/etc/systemd/journald.conf` to `Storage=persistent`
2. Reboot your device (`reboot`) or restart the logger system (`systemctl restart systemd-journald`)

NOTE: Don't forget to clean the logs when you are switching from persistent to volatile storage. More details see section *Maintaining the Logs* below.

26.4 Maintaining the Logs

In embedded devices the disk usage might be essential. The logger system provides several tools to maintain the logs.

26.4.1 Disk Usage

To check the amount of disk space the logs are taking, just run this command:

```
root@am335x-nmhw21:~# journalctl --disk-usage
Archived and active journals take up 18.0M in the file system.
```

26.4.2 Cleaning

There are several methods provided to clean the logs.

Cleaning your logs to a specific size run `journalctl --vacuum-size=`. This removes the oldest archived journal files until the disk space they use falls below the specified size (specified with the usual "K", "M", "G" and "T" suffixes):

```
root@am335x-nmhw21:~# journalctl --vacuum-size=4M
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-00000000000014c8-0005ae5e2a594f10.journal␣
↪(2.0M).
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-0000000000001af7-0005ae5ecf0899e3.journal␣
↪(2.0M).
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-00000000000020d0-0005ae5f68b8138f.journal␣
↪(2.0M).
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-00000000000026c9-0005ae600002f7a6.journal␣
↪(2.0M).
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-0000000000002cc1-0005ae60973a5468.journal␣
↪(2.0M).
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-00000000000032b9-0005ae612e720fcf.journal␣
↪(2.0M).
Vacuuming done, freed 12.0M of archived journals from /run/log/journal/
↪c8b8c280f0bc43aba10c21e3574e81fc.
```

Cleaning your logs using a specific time run `journalctl --vacuum-time=`. This removes all archived journal files contain no data older than the specified timespan (specified with the usual "s", "m", "h", "days", "months", "weeks" and "years" suffixes):

```
root@am335x-nmhw21:~# journalctl --vacuum-time=1s
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-00000000000038b1-0005ae61c5b11a9d.journal␣
↪(2.0M).
Deleted archived journal /run/log/journal/c8b8c280f0bc43aba10c21e3574e81fc/
↪system@335f2d7f78764e27ba522dc69770346f-0000000000003ef0-0005ae6259e61874.journal␣
↪(2.0M).
Vacuuming done, freed 4.0M of archived journals from /run/log/journal/
↪c8b8c280f0bc43aba10c21e3574e81fc.
```

NOTE: When switched from persistent to volatile and after cleaning, it might also be possible to remove the persistent logs by calling `rm -rf /var/log/journal/*`.

CHAPTER 27

NetModule Linux Networking

27.1 NetworkManager

27.1.1 nmcli

- `nmcli c edit ethernet` - edit the ethernet connection interface.
- `nmcli c modify ethernet ipv4.method auto` - oneline edit, useful for scripts

CHAPTER 28

Power Management

28.1 Overview

NetModule OEM Linux Distribution provides two standard high-level power management strategies:

- system-wide power management,
- working-state power management.

System-wide power management strategy is using global low-power states to reduce system activity. In these states, referred as sleep states, user space code cannot be executed. Depending on sleep state supported by the platform, different levels of energy saving can be achieved. To get back to the working state, the system expects to receive a special signal from one of the designated devices.

Working-state power management strategy corresponds to adjusting the power states of individual hardware components.

28.1.1 Supported hardware

- DA9063 System PMIC

28.1.2 Supported sleep states

Standby This state provides a relatively straightforward transition back to the working state. In this state the system core logic retains power and no operating state is lost. It offers moderate, real energy savings.

PMIC's Power Down Mode Platforms with DA9063 System PMIC can support more sophisticated energy saving options by disabling the system power domain. The system power domain is enabled on a signal received from a preconfigured wake-up device.

28.1.3 Supported wake-up scenarios

- RTC based alarm
- IMU based events: single-tap, double-tap
- ONKEY event
- KL15 event

28.2 Smart Battery

NetModule OEM Linux Distribution can run on devices supplied by a smart battery. To get current status of the battery our distribution provides a battery test tool.

28.2.1 User space tools and configuration

Enter standby state:

```
$ echo standby > /sys/power/state
```

Wake up from standby after 10 seconds:

```
rtcwake -d /dev/rtc0 -m standby -s 10
```

or without rtcwake in your image:

```
echo +10 > /sys/class/rtc/rtc0/wakealarm && echo standby > /sys/power/state
```

Enter PMIC's power down mode:

```
$ poweroff
```

Start system after ~1 min when in PMIC's power down mode:

```
echo +60 > /sys/class/rtc/rtc0/wakealarm && poweroff
```

Using a battery test:

```
$ batterytest -h
battery tool

Examples:
  batterytest -v         get voltage
  batterytest -a         get current
  batterytest -t         do battery test

Main modes of operation:
 -h, --help              print this help
 -A, --all               print all battery information
 -v, --voltage           get battery voltage
 -a, --current           get battery current
 -c, --capacity          get remaining capacity
 -r, --reg               raw mode, register access
 -t, --test              run battery test
```

CHAPTER 29

Remote GPIO driver

29.1 Configuration

By default, driver will create 32 GPIO's (named RGPIO0-RGPIO31), and will set server IP and port as 192.168.1.42:6666.

It is possible, however, to configure Remote GPIO driver via device tree:

```
remote-gpios {
  compatible = "remote-gpios";

  ip = "192.168.1.64";
  port = /bits/ 16 <6666>;

  remote-gpio@0 {
    label = "umgpo0";
  };
  remote-gpio@1 {
    label = "umgpo1";
  };
  remote-gpio@2 {
    label = "umgpo2";
  };
  remote-gpio@3 {
    label = "umgpo3";
  };

  remote-gpio@4 {
    label = "umpu0";
  };
  remote-gpio@5 {
    label = "umpu1";
  };
  remote-gpio@6 {
```

(continues on next page)

(continued from previous page)

```
    label = "umpu2";
  };
  remote-gpio@7 {
    label = "umpu3";
  };

};
```

The ip-address and port can be changed via SysFS under `/sys/class/remote-gpio/remote-gpio/config/`. The service "um-service-config.service" will configure the port to either "6666" or "7020" depending on the user-module firmware revision.

29.2 Protocol

The Protocol uses TCP. It consists of *commands* and *events*, where *command* is something that is received by user module (server), and *event* is a state notification to the client. There can be multiple *events* or *commands* in one TCP-Package.

- **Command: set GPIO output**

```
O<id><state>
```

Where:

- *id* is two digit pin hex number
- *state* is 0 for low, 1 for high

Examples:

- Set GPIO 13 low

```
O0D0
```

- Set GPIO 8 high

```
O081
```

- **Event: GPIO input state**

```
I<id><state>
```

Where:

- *id* is two digit pin hex number
- *state* is 0 for low, 1 for high

Examples:

- GPIO input 2 set to low

```
I020
```

- GPIO input 11 set to high

```
I0B1
```

- GPIO output 1 and 2 are set to high

```
O011O021
```

29.3 Usage

Since linux 4.8 the GPIO sysfs interface is deprecated. User space should use the character device instead. Current de-facto standard of using new GPIO system is libgpiod library and its tools:

- gpiodetect
- gpioinfo
- gpioget
- gpioset
- gpiofind
- gpiomon

CHAPTER 30

System Clock / Date Time

There might be several time giving sources like GNSS, NTP, etc that can be used to synchronize the system clock and other clock dependent parts.

30.1 Synchronization

Chrony is able to perform such clock synchronizations. For further information please have a look at *Chrony*. Chrony is also able to synchronize the HW clock (RTC) automatically if it drifts away.

Another tool to synchronize/handle/update the HW clock (RTC) is `timedatectl` which competes against chrony. **This means be cautious when synchronizing/updating the HW clock manually**

30.2 HW Clock Synchronization/Update

As described in section above the RTC can be updated using different tools. The following sections show how this can be performed.

30.2.1 timedatectl

NOTE: Using `timedatectl` you are able to update the HW clock manually.

If no time source is present no update takes place automatically. Then a manual update is possible with:

```
timedatectl set-local-rtc n
```

Please read also the timedatectl man page when using `timedatectl`.

CHAPTER 31

Wi-Fi

31.1 Overview

- Used WIFI (and BT) Chip: WL1837MOD
- linux-firmware repository: `git://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git`
- Commit id: 4c0bf113a55975d702673e57c5542f150807ad66
- Which is basically: wl18xx: update firmware file 8.9.0.0.76
- NetworkManager (nmcli) connection configuration files are stored under `/etc/NetworkManager/system-connections`

31.1.1 802-11 Standard

- a: 5 GHz-Band, up to 54 Mbits/s
- b: 2.4 GHz-Band, up to 11 Mbits/s
- g: 2.4 GHz-Band, up to 54 Mbits/s
- n: 2.4 & 5 GHz-Band, up to 600 Mbits/s
- ac: 5 GHz-Band, up to 1.3 Gbits/s

31.2 Client Mode

Scan and connect to existing access point (AP)

```
$ nmcli d wifi list
$ nmcli d wifi connect <MYWLAN> password <my_password>
```

31.3 Access Point Mode

Note: Set region code to get right channels!

31.3.1 Create Access Point in 2.4 GHz Band

Create hotspot with SSID Hostspot24, no encryption

```
$ nmcli con add type wifi ifname wlan0 con-name Hostspot24 autoconnect yes ssid␣
↪Hostspot24
$ nmcli con modify Hostspot24 802-11-wireless.mode ap 802-11-wireless.band bg ipv4.
↪method shared
$ nmcli con up Hostspot24
```

Create hotspot with SSID SecSpot, wpa-psk encryption with password: 12345678

Note: for wpa-psk encryption password has to be 8 characters or more.

```
$ nmcli con add type wifi ifname wlan0 con-name SecSpot autoconnect yes ssid SecSpot
$ nmcli con modify SecSpot 802-11-wireless.mode ap 802-11-wireless.band bg ipv4.
↪method shared
$ nmcli con modify SecSpot wifi-sec.key-mgmt wpa-psk
$ nmcli con modify SecSpot wifi-sec.psk "12345678"
$ nmcli con up SecSpot
```

31.3.2 Create Access Point in 5 GHz Band

Create hotspot with SSID Hostspot50, no encryption

```
$ nmcli con add type wifi ifname wlan0 con-name Hostspot50 autoconnect yes ssid␣
↪Hostspot50
$ nmcli con modify Hostspot50 802-11-wireless.mode ap 802-11-wireless.band a ipv4.
↪method shared
$ nmcli con up Hostspot50
```

Create hotspot with SSID SecSpot5, wpa-psk encryption with password: 12345678

Note: for wpa-psk encryption password has to be 8 characters or more.

```
$ nmcli con add type wifi ifname wlan0 con-name SecSpot5 autoconnect yes ssid SecSpot5
$ nmcli con modify SecSpot5 802-11-wireless.mode ap 802-11-wireless.band a ipv4.
↪method shared
$ nmcli con modify SecSpot5 wifi-sec.key-mgmt wpa-psk
$ nmcli con modify SecSpot5 wifi-sec.psk "12345678"
$ nmcli con up SecSpot5
```

CHAPTER 32

WWAN

32.1 Overview

- WWAN modules : TOBY - L210
- Modem Firmware : 16.19 (Caution: older versions may not work stable!)
- NetworkManager (nmcli) connection configuration files are stored under `/etc/NetworkManager/system-connections`

32.2 Preparation

1. Insert SIM Card in sim slot

32.3 Usage

All the configuration can be done by NetworkManager but sometimes it can useful to check lower level configurations with ModemManager

32.3.1 Firmware update

On some devices, the modem is delivered with the firwmare 15.63. It is highly recommended to upgrade the firmware to most recent version 16.19.

How to check firmare version and upgrade is explain at *Updating firmware*.

32.3.2 Initial configuration

The WWAN modem is configured at each boot by a script named wwan-config. This script is using the configuration file */etc/wwan/wwan0* to setup the modem before letting ModemManager handle it.

This configuration file is divided in three sections:

- apn: must configured to use a private APN
- sim: is used to choose between the different sim cards available on the device
- ublox: low level configurations for ublox modem. This should normaly not be modified

APN configuration

When using a private APN, this section has to be configured with the following fields:

```
[apn]
apn=<APN>
user=<USER>
password=<PASSWORD>
```

When the default APN provided by the network when using LTE must be used, make sure that this fields are not set.

After any change to this file, the system has to reooted or the following command to be run:

```
$ systemctl restart wwan-config@wwan0
```

SIM card configuration

This section is used to choose which SIM card to use with the modem. There are four SIM cards slot that can be used by the modem.

```
[sim]
SIM=<value>
```

Where <value> can be :

- auto: The script will detect if a physical SIM card is present and switch to m2m SIM card (soldered to the board) if it is not the case
- sim1: Use the physical SIM card on the main board
- m2m: Use the m2m SIM card soldered on the main board
- ui-top: Use the SIM card that is on top of the User Interface
- ui-btm: Use the SIM card that is on the bottom of the User Interface

After any change to this file, the system has to be rebooted or the following command to be run:

```
$ systemctl restart wwan-config@wwan0
```

32.3.3 NetworkManager commands

```
$ # Create connection
$ nmcli c add type gsm con-name wwan ifname "" ipv6.method ignore gsm.apn <APN>

$ # Create connection with APN authentication
$ nmcli c add type gsm con-name wwan ifname "" ipv6.method ignore \
  gsm.apn <APN> gsm.username <USER> gsm.password <PASSWORD>

$ # Set PIN number
$ nmcli c modify wwan gsm.pin <pin number>

$ # Start the connection
$ nmcli c up wwan
```

32.3.4 ModemManager configuration

```
$ mmcli -L  # list modems and get modem id
$ mmcli -M  # list modems in a loop, useful when waiting after a reset
$ mmcli -m 0 # See state of the modem 0
$ mmcli -i 0 --pin=<pin number>  # Entering pin on modem 0
$ mmcli -m 0 -r # Reset the modem
```

32.3.5 Low level configuration

1. Launch ModemManager in debug mode

```
$ systemctl stop ModemManager
$ ModemManager --debug > /dev/null 2> /dev/null &
```

2. Execute the commands

```
$ mmcli -m 0 --command "AT+UBMCONF=2"  # Set modem to bridge mode
$ mmcli -m 0 --command 'AT+UUSBCONF=2,"ECM",0'  # Set USB mode to ECM
$ mmcli -m 0 --command 'AT+UUSBCONF=3'  # Set USB mode to RNDIS
```

3. Restart ModemManager normaly

```
$ killall ModemManager
$ systemctl start ModemManager
```

32.3.6 ModemManager extensions

The ModemManager version used in NetModule linux is the version 1.10 with some NetModule specific extensions.

It is maintained and up to date with latest bug and security fixes.

NetModule did the following changes to the community version :

1. Support configuration of default EPS bearer for u-blox modems

In 4G (LTE), the handling of the APN configuration is different than in 2G and 3G and specific to each vendor. U-blox modems is not supported in the community version.

2. Handling of reconnect requests

Events like reconnection and disconnection on the radio side trigger AT messages that are not handled by ModemManager 1.10. The NetModule version handles this messages, leading to a faster reconnection.

3. Change AT commands timeout to 3 minutes

The u-blox modems can take up to 3 minutes before answering and AT command. The default timeout varies between 3 and 60 seconds depending on the commands. This difference made MM send more while modem was still processing the first one, leading to a lock of the modem.

CHAPTER 33

Booting with custom Linux kernel or ramdisk

If it's necessary, it is possible to boot custom a Linux kernel with existing OSTree controlled file system.

33.1 Provisioning over tftp

1. Load the ostree necessary variables.

```
run bootcmd_otenv
```

2. Load your own kernel.

```
tftp $kernel_addr_r fitImage
```

xor load the installed kernel:

```
ext4load mmc 1:1 $kernel_addr_r /boot$kernel_image
```

3. Load your own ramdisk.

```
tftp $ramdisk_addr_r ramdisk
```

xor load the installed ramdisk:

```
`ext4load mmc 1:1 \$ramdisk_addr_r /boot\$ramdisk_image;
```

4. Boot the system.

```
bootm $kernel_addr_r $ramdisk_addr_r
```

33.2 Provisioning over USB

1. Initialize USB

```
usb reset
```

2. Load the ostree necessary variables.

```
run bootcmd_otenv
```

3. Load your own kernel.

```
fatload usb 0:1 $kernel_addr_r fitImage
```

xor load the installed kernel:

```
ext4load mmc 1:1 $kernel_addr_r /boot$kernel_image
```

4. Load your own ramdisk.

```
fatload usb 0:1 $ramdisk_addr_r ramdisk
```

xor load the installed ramdisk:

```
ext4load mmc 1:1 $ramdisk_addr_r /boot$ramdisk_image;
```

5. Boot the system.

```
bootm $kernel_addr_r $ramdisk_addr_r
```

CHAPTER 34

Create a fitImage

To create a fitImage you need a .its file. Luckily YoctoProject creates this file for us. You find in the deployed image folder (e.g. `hancock-os/shared-build/tmp/deploy/images/am335x-nmhw21/fitImage-its-am335x-nmhw21.its`).

In this file, you will find this line in the kernel section:

```
data = /incbin/("linux.bin");
```

This tells us what the filename of your legacy kernel (zImage, uImage, Image) needs to be called.

You will also find this line in the dtb section:

```
data = /incbin/("arch/arm/boot/dts/am335x-nmhw21-prod1.dtb");
```

This tells us where the the dtb is expected.

34.1 Steps (nmhw21)

1. Create a folder structure that looks like this:

```
|-- fitImage-its-am335x-nmhw21.its
|
|-- linux.bin ( --> your kernel image)
|
|-- arch/arm/boot/dts/am335x-nmhw21-prod1.dtb
```

2. To create the fitimage now run in your folder:

```
mkimage -f fitImage-its-am335x-nmhw21.its fitImage
```

CHAPTER 35

Indices and tables

- genindex
- modindex
- search