

# Running a standard Linux distribution on a smartphone

**Masterthesis  
Computer Science**

July 13, 2015

Sebastian Reichel (Matrikelnummer: 1104957)  
Otto-Wels-Str. 82  
26133 Oldenburg

**Erstprüfer  
Zweitprüfer**

Prof. Dr.-Ing. Oliver Theel  
Eike Möhlmann, Dipl. Inform.



## **Erklärung zur Urheberschaft**

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Oldenburg, den July 12, 2015

---

Sebastian Reichel

# Contents

<b>1. Introduction</b>	<b>2</b>
<b>2. Hardware Architecture</b>	<b>6</b>
<b>3. Serial Access</b>	<b>12</b>
<b>4. Linux Kernel</b>	<b>23</b>
4.1. N900 kernel support . . . . .	25
4.2. Accelerometer . . . . .	27
4.3. Modem . . . . .	29
4.4. ARM Errata 430973 . . . . .	32
4.5. Bluetooth . . . . .	33
4.6. Camera Subsystem . . . . .	36
<b>5. Userland</b>	<b>41</b>
5.1. Graphical User Interface . . . . .	41
5.2. Wireless LAN . . . . .	42
5.3. The modem . . . . .	43
<b>6. Function tests</b>	<b>47</b>
6.1. Updates & Security Updates . . . . .	47
6.2. Voice Calls . . . . .	47
6.3. Short Messages . . . . .	48
6.4. Web Surfing . . . . .	49
6.5. Customization . . . . .	50
<b>7. Outlook</b>	<b>51</b>
<b>8. Conclusion</b>	<b>55</b>

<b>Appendix</b>	<b>56</b>
<b>A. Debian Image</b>	<b>57</b>
<b>B. ISI-Protocol</b>	<b>62</b>
<b>C. Accelerometer Patches</b>	<b>70</b>
<b>D. OpenSCAD code</b>	<b>75</b>
<b>E. Bill of Materials</b>	<b>76</b>
<b>F. Simple Ofono Dialer</b>	<b>77</b>
<b>Bibliography</b>	<b>83</b>
<b>Glossary</b>	<b>86</b>

## List of Figures

2.1. N900 DT graph . . . . .	9
2.2. DT graph of OMAP3430's system bus . . . . .	10
2.3. DT graph Nokia N900's second I2C port . . . . .	10
3.1. Coverless N900's back with labeled serial test pads . . . . .	12
3.2. Pogo-Pin profile . . . . .	13
3.3. Battery adapter design in Inkscape . . . . .	14
3.4. Serial adapter schematic (designed with KiCad) . . . . .	16
3.5. Generating G-Code for Gerber-files using FlatCAM . . . . .	18
3.6. left: Front and back of the adapter's PCB; right: PCB with crepe-tape-stencil . . . . .	18
3.7. Colorized vector information for the laser cutter . . . . .	19
3.8. 3D-rendering of the layers . . . . .	20
3.9. N900 with attached debug adapter . . . . .	21
4.1. Flow-graph of kernel forks . . . . .	23
4.2. Kernel-development flow . . . . .	24
4.4. Modem interconnection . . . . .	29
4.5. Modem protocol stack . . . . .	31
4.6. BCM2048 setup . . . . .	34
4.7. Camera setup . . . . .	37
4.8. OMAP's image signal processor in Linux . . . . .	39
4.9. N900 camera setup in Linux . . . . .	40
5.1. Screenshot from Enlightenment running in vertical split mode . . . . .	43
6.1. Voice call handling in ofono-dialer . . . . .	48
6.2. SMS handling in ofono-dialer . . . . .	49
6.3. LXDE running on the Nokia N900 . . . . .	50
B.1. GPS Session . . . . .	64
B.2. GPS Session in Wireshark . . . . .	69

## List of Tables

1.1. Smartphone Operating System Comparison . . . . .	4
4.1. Major Linux Kernel Releases . . . . .	25
4.2. N900 Linux Kernel Support (Status: Linux Kernel 4.0) . . . . .	26
4.3. Nokia H4+ Packet Types . . . . .	34
B.1. PhoNet Header . . . . .	62
B.2. ISI Resource IDs (excerpt of most important IDs) . . . . .	62
B.3. ISI Header . . . . .	63
B.4. GPS Session: Status Indication Subscription . . . . .	63
B.5. GPS Session: Hybrid Tracking Request . . . . .	65
B.6. GPS Session: Hybrid Tracking Response . . . . .	65
B.7. GPS Session: Socket Open Request . . . . .	66
B.8. GPS Session: Socket Send Request . . . . .	66
B.9. GPS Session: Socket Receive Request . . . . .	67
B.10. GPS Session: GPS Status Indication: No Lock . . . . .	67
B.11. GPS Session: GPS Status Indication: Lock Acquired . . . . .	67
B.12. GPS Session: Hybrid Location Data . . . . .	68
E.1. Bill of Materials for UART Adapter PCB . . . . .	76

## Listings

2.1. Simplified Nokia N900 Device Tree Source File . . . . .	7
2.2. Simplified and Merged SPI4 Excerpt from Nokia N900 Device Tree Source File . . . . .	8
3.1. Nokia N900 Boot Log . . . . .	21
4.1. Simplified Nokia N900 Display Connection DT excerpt . . . . .	37
5.1. Starting X-Server together with an XTerm . . . . .	41
5.2. X.org Touchscreen Configuration . . . . .	42
5.3. Patch adding ofono support to the cmt-speech plugin of the FSO-audio daemon . . . . .	45
6.1. Initialize a voice call from command line using mdbus2 . . . . .	47
6.2. Send a SMS using mdbus . . . . .	48
C.1. [PATCH] lis3lv02d: DT: use s32 to support negative values . . . . .	70
C.2. [PATCH] lis3lv02d: DT: add wakeup unit 2 and wakeup threshold . . . . .	71
C.3. [PATCH] Documentation: DT: lis302: update wakeup binding . . . . .	72
C.4. [PATCH] DTS: ARM: OMAP3-N900: Add lis3lv02d support . . . . .	73
D.1. OpenSCAD-Code for 3D-rendering of the Serial-Adapter . . . . .	75
F.1. Simple Ofono dialing and short message application . . . . .	77



**Abstract** The aim of this thesis is running a free operating system on a smartphone, which shares many components and principles with the operating system already used on the desktop PC. As a result users and developers can use the software they are used to on their PC and on their smartphone. Compared to other closed source alternatives it also means, that any code audits are more likely reducing the probability of built-in backdoors.

The thesis is written using Debian as example Linux distribution and uses the Nokia N900 as hardware base. One reason for choosing the Nokia N900 was, that the main Linux kernel used by the Debian project has already support for all base hardware components. The first part of the thesis will describe how the kernel support for the Nokia N900 can be improved, while the second part focuses on the userspace software.

# 1. Introduction

Nowadays it is common, that one can install a arbitrary [Linux distribution](#) on ones notebook or desktop computer. Installing it on a smartphone is a completely different issue, though. Currently the [Linux kernel](#), as available on [kernel.org](http://kernel.org), does not support a complete smartphone and as a result most [Linux distributions](#) do not provide any smartphone support.

Smartphones, in difference to desktop PCs and notebooks are commonly shipped with a locked down system running [Android](#), iOS or Windows Phone these days [27]. Of these smartphone distributions only [Android](#) is mostly based on open source software, while iOS and Windows Phone are closed source software and always shipped with the hardware. From a users point of view the open source development model has a few advantages:

1. **Auditability:** Since the software's source code is available in public, it is possible to review it. If the user does not understand the programming language, there is still an increased chance, that other people review the source code. This is an important factor in times of intelligence agencies being suspected of adding backdoors to software.
2. **Freedom:** Since everything is open, everything can be adjusted to fit the user's preferences. This also counts for any bugs, that are found. While closed source software requires, that the vendor fixes the bug, user's can do it themselves. There is no dependency on the vendor's support.

Additionally trying the software is most times free of charge. While the term free in free software refers to the independence, most software is also free of charge.

Since Windows Phone and iOS are not open source software and may even come with a hardware lock, they will not be further compared with the aimed smartphone support for a major [Linux distribution](#). [Android](#)'s base system on the other hand is available as open source software and is even using the Linux kernel as base. Nevertheless [Android](#) is quite different from most other [Linux distributions](#), since it brings its own software stack on top of the Linux kernel with minimal use of existing projects already available. Apart from that vendors are not actively working on getting their software modifications back into the original projects. Instead many vendors copy the software's source code and continue working on the copy (also known as [fork](#)). In the Android community these kind of forks are usually referred to as „vendor-customized“.

As a result of the missing work on getting the software modifications back into the original projects, the current Linux kernel does not support the complete hardware components of any smartphone. At the same time the amount of work required to integrate the vendor customizations increases with time, since the [Linux kernels](#) changes. The task of integrating vendor drivers to a newer kernel with the aim of including them in the main Linux kernel tree, will be described more detailed in [chapter 4](#).

Apart from [Android](#) there are a few other related projects available on the market:

**OpenMoko:** One of the first projects trying to run a Linux kernel on a phone is [OpenMoko Linux](#), which was worked on by [OpenMoko Inc.](#) from 2007-2009. It's a custom Linux distribution created for their

phone, the Neo FreeRunner. Their distribution was based on the Ångström distribution and uses the Linux kernel, a GNU base system as well as the X.org server. The [Graphical User Interface \(GUI\)](#) parts were custom developments, but based on the same toolkits as used in other [Linux distributions](#): GTK+, Qt and [Enlightenment Foundation Libraries \(EFL\)](#). The user base decreased over the time, though. One reason may be the outdated hardware (400 MHz ARMv4 CPU and 128MB system memory).

However the project resulted in the development of a smartphone middleware, which abstracts hardware and makes the creation of a hardware independent [GUI](#) much easier [9]. Later a new distribution named [Stable Hybrid Release \(SHR\)](#) has been created on top of the Ångström distribution reusing the smartphone middleware developed previously [35]. The custom software from the [SHR](#) and the [freesmartphone.org \(FSO\)](#) stack has been ported to a few other distributions including [Debian](#). In opposit to previous projects, the [FSO](#) project took care of proper device abstraction and provides a full framework for different smart-phone related tasks.

**Maemo & MeeGo:** Independently from the [OpenMoko](#) related projects [Nokia](#) released a internet tablet in 2005 [29], which was dependent on WLAN and did not contain a mobile broadband unit. It was released with a [Debian](#) based distribution called [Maemo](#), though. Later, in 2009 [Nokia](#) released an updated variant of the internet tablet, the [N900](#), which did include a cellular broadband modem. Together with the new hardware a new version of [Maemo](#) was released, which did contain software for dialing and short messages, so that the original Internet tablet series became a smartphone series.

After the [N900](#) was released the market share of different phone vendors changed noticeably, resulting in Android and iOS representing almost the complete market in most countries. Nokia became much less relevant, since their newer phones are based on Microsoft's Windows operating system. Not least because of the internal changes at Nokia, the N900 has not been supported long. Since it has been developed with many bits being open source the community has taken over support though and released (security) updates for the system. This task becomes harder with time though, since the source code of some applications and middle layers is not available making it hard to update some libraries.

**Sailfish OS:** After [Nokia](#) stopped working on [Maemo](#) (and its successor MeeGo) and cooperated with [Microsoft](#) to use Windows Phone more extensively, some of the original developers left [Nokia](#) to found a new company called Jolla. They continued working on the distribution under the name Sailfish OS. Sailfish OS in opposit to [Maemo](#) is still actively been worked on and gets regular security updates. Besides that it has an [Android](#) compatibility layer, so that most applications written for [Android](#) can also be used with Sailfish OS. At the moment some parts of the operating system are not open source, though.

**Neo900:** Another project, which tried to hop on the [N900](#)'s success in the Open Source community is called Neo900 [26]. It has been started by a small company in Munich, which is working on a replacement board for the [N900](#)'s case. This board is supposed to provide all of the original features, but with an upgraded CPU, more memory and an open hardware approach. The same company also provided a replacement board for the [OpenMoko](#) Freerunner and thus this project merges the former OpenMoko community with the [Maemo/N900](#) community. The hardware has not yet been released and thus this project will not be in the focus of this thesis.

**Ubuntu Touch:** Another related work currently being worked on by Canonical is Ubuntu Touch, which is developed as open source software. This project is quite near to this thesis' goals, but has only limited availability. The project relies on existing software for the phone stack and reuses the [Android](#) compatibility layer developed for Sailfish OS. It does not make use of standard kernels at the moment, though. Instead

## 1. Introduction

the CyanogenMod's [Android](#) kernel and some related low level services are reused with older releases even being started in a [chroot](#) [15, 1].

**Android chroot:** Last but not least there is the possibility of starting a [chroot](#) with one of the existing Linux distributions from [Android](#). This option works with most phones and needs only minimal modification, but has the disadvantage of high memory demands and bad integration. For example its problematic to access the phone hardware from the [chroot](#). Apart from that one is still dependent on the underlying [Android](#) and thus from the phone vendor regarding security (and general) updates.

**Summary:** [Table 1.1](#) gives an overview about existing operating systems currently running on smartphones compared to the aim of this thesis. Since the mainline kernel, which is used as basis by the major [Linux distributions](#), does not provide support for a complete smartphone, none of the existing smartphone operating systems are using it. This is a problem for supporting a phone with the common [Linux distributions](#), which try to avoid customized kernels for different reasons (more on that in [chapter 4](#)). Thus the thesis will start with adding support for a smartphone to the Linux kernel as available from `kernel.org`.

For this task the thesis will use the Nokia N900 as basis, for the following reasons:

1. The Nokia N900 has quite good support in the main kernel tree.
2. The phone has a hardware keyboard reducing the amount of work required to port some applications, since no virtual keyboard is needed.
3. The bootloader is not locked, so installing ones own kernel is easily possible.
4. The phone has a microSD card slot making it easy to add ones own operating system without overwriting the existing one. It also makes it possible to debug the userland on a PC using an emulator.
5. It's possible to reach a serial port for kernel development.

If the kernel is compiled with the right options, it will already run on the Nokia N900. Once the kernel support has been sufficiently enhanced, Debian will be used as base operating system. Debian already contains the [FSO](#) software stack as well as the graphical applications from the [SHR](#) distribution. In opposit to other distributions it also tries to keep its main archive free of any closed source software components.

The next chapter has a look at the N900's hardware and which hardware components must be supported to get the phone running a basic Linux distribution. It will also be analysed what components will be needed to develop for the phone. Then in the third chapter, it will be analysed how serial access to the Nokia N900 can be established. This chapter will describe the construction of an adapter to access the serial pins, which are located under the phone's battery compartment. In the fourth chapter a couple of drivers for important unsupported hardware components, like modem speech data support, are integrated into the current mainline kernel. With basic kernel support established the fifth chapter describes the required

	open source	kernel	Android compatibility	userland software
Windows Phone	No	Windows	No	custom
iOS	No	iOS	No	custom
Android	Partially	Android	-	custom
OpenMoko	Yes	custom	No	GNU based
Maemo	Partially	custom	No	GNU based
MeeGo	Partially	custom	No	GNU based
SailfishOS	Partially	Android	Yes	MeeGo software stack
Ubuntu Phone	Yes	Android	Yes	GNU based
chroot	Partially	Android	-	-
thesis aim	Yes	mainline	nice to have	GNU based

Table 1.1.: Smartphone Operating System Comparison

userspace software, that acts as a middle layer between the Linux kernel and the user software. It will also highlight a few user applications, that are available for usage on the phone. Then, in the sixth chapter, the previously described software stack will be tested for functionality and usability followed by a chapter, which summarizes all open tasks identified during the thesis categorized according to the related chapter. Finally the last chapter contains a conclusion.

Some additional content is provided in the appendix, which contains a a step-by-step tutorial to create a Debian SD-card image for the Nokia N900 in [Appendix A](#). Then [Appendix B](#) describes the [Intelligent Service Interface \(ISI\)](#) protocol, which is used for communication between the N900's modem and the processor. Next in [Appendix C](#) the kernel patches required to add support for the N900's accelerometer are included. Then, [Appendix D](#) provides details about generating a 3D model of the serial adapter built in [chapter 3](#). The following [Appendix E](#) contains an overview of all components required to built the [Printed Circuit Board \(PCB\)](#) for the UART adapter described in [chapter 3](#). Last but not least [Appendix F](#) contains the source code for a proof of concept dialer used in [chapter 6](#).

The next chapter will describe the [N900's](#) hardware architecture and how its abstracted by the Linux kernel.

## 2. Hardware Architecture

This chapter will describe the hardware architecture of the Nokia N900 as base for the development of Linux kernel drivers, which follows in the next chapter.

The smartphone is based on an [OMAP-3430 System on a Chip \(SoC\)](#) from [Texas Instruments](#), which contains a Cortex-A8 [Central Processing Unit \(CPU\)](#). The kernel support for most [ARM](#) platforms is currently being converted to [device tree \(DT\)](#) based booting. The [DT](#) is a data structure, that has been developed for usage with Open Firmware to inform the operating system about any hardware, that can not identify itself. Since the ARM platform does not use any specific firmware interface, most operating systems encoded any knowledge about connected peripherals in their source code. For the Linux kernel, which supports a many different boards this became a maintenance problem, so it has been decided to remove the board specific code and load the board information from via [DT](#) instead. This is an ongoing process, which is far from complete, but the [OMAP](#) subsystem has already dropped support for non-[DT](#) based booting for newer chip revisions. While the Linux 4.1 kernel still supports non-[DT](#) based booting for the Nokia N900, the subsystem maintainer plans to remove support for all OMAP3 based boards in Linux 4.3 [20]. More details about the kernels movement to [DT](#) will follow in [chapter 4](#). But this chapter will already make use of the abstract hardware information provided by the kernel's [DT](#) file.

The [DT](#) file for the Nokia N900 can be found in the Linux kernel's source code<sup>1</sup> in `arch/arm/boot/dts/omap3-n900.dts` and contains information about interconnections between different hardware components. A simplified version (the whole file including the SoC information is over 4000 lines in the Linux 4.1 kernel) can be seen in [Listing 2.1](#).

The first line starts with the inclusion of another [DT](#) file, which provides information about all hardware information, which are part of the OMAP3430 [SoC](#). For example it contains information about the ARM Cortex-A8 processor core, the system bus and the modules and any modules, that are part of the processor.

In the third line, the scope is changed to the root node `/`, so that all content between line 3 and line 16 is are children of the root node. Next in line 4, the compatible property of the root node is set. The compatible property is a standard property defined in the ePAPR standard<sup>2</sup>, which identifies the device. The standard requires, that the value is a list of strings describing the device, which are ordered from most specific to most general. The format is usually in the form "manufacturer,product".

While most information about the N900's CPU is actually part of the OMAP3430 description included in the first line, it cannot describe where the CPU gets its energy from (since that part is no [SoC](#) specific, but board specific). Thus the lines 6 and 7 are used to change the scope to the first CPU's node (`cpus/cpu@0`), so that the information about the power supply source can be added. Since the N900 is a single-core device, no other CPU's must be referenced. The CPU's power supply source is specified by referencing the voltage regulator in line 8 using the `&` character. This kind of reference is commonly used to describe connections between different devices or modules, such as an interrupt line or a [Direct Memory Access \(DMA\)](#) channel.

---

<sup>1</sup>The Linux kernel's source code is available on [kernel.org](http://kernel.org)

<sup>2</sup><https://www.power.org/documentation/epapr-version-1-1/>

Next from line 12 to line 15 the phone's memory is described by adding a new node named memory. It contains information about the memory position on the system-bus (0x80000000) and its size (0x10000000 = 256MB).

Listing 2.1: Simplified Nokia N900 Device Tree Source File

```
1 #include "omap34xx-hs.dtsi"
2
3 / {
4     compatible = "nokia,omap3-n900", "ti,omap3430", "ti,omap3";
5
6     cpus {
7         cpu@0 {
8             cpu0-supply = <&vcc>;
9         };
10    };
11
12    memory {
13        device_type = "memory";
14        reg = <0x80000000 0x10000000>; /* 256 MB */
15    };
16 };
```

Listing 2.2 shows another extract of the Nokia N900's DT file describing the connection of an optional hardware component. It describes how the N900's [Wireless Local Area Network \(WLAN\)](#) chip is connected to the system via one of the [OMAP's SPI<sup>3</sup>](#) ports.

The listing starts with a reference to the system-bus (line 1), which is extended by a node describing the fourth [SPI IP-Core](#) (line 2). This controller can be identified using the compatible property (line 3) and mapped to the correct system-bus address space using the information provided by the *reg* property (line 4). The *reg* property is interpreted the same way as it was interpreted for the memory node in Listing 2.1: The [SPI](#) module has a base address of *0x480ba000* and has a size of 256 Bytes.

The next two properties (interrupt-parent and interrupts in line 5 and 6) describe which interrupts are connected to the [SPI](#) controller. Neither the data, nor the standard define the interrupts usage, so that part must be known by the driver.

Similarly the next two properties (*dmass* and *dma-names* in line 7 and 8) are used for supplying [DMA](#) information. In opposit to the interrupt description the *dmass* property references the [DMA](#) controller and the used channel and additional information about the channels is provided using the *dma-names* property.

The *pinctrl* properties in line 9 and 10 are required for setting up the [SPI](#) pins into [SPI](#) mode instead of using them e.g. as simple [General Purpose Input/Output \(GPIO\)](#). The referenced *mcspi4\_pins* provides all information required to mux the pins correctly.

Last but not least the properties in line 12 and 13 describe how the children's *reg* property's value should be parsed. In this case it defines, that one entry in the *reg* property consists of one address value and zero size values. Providing this makes it possible to automatically parse the children's *reg* properties. In case of [SPI](#) the *reg* property is used to provide the [SPI](#) address of a device.

The properties are followed by a sub-node describing the w1251 [WLAN](#) chip, which is connected to the

---

<sup>3</sup>a serial interface for short-distance inter-chip communication

## 2. Hardware Architecture

fourth **SPI** bus. It again contains a compatible node, so that the hardware can be identified. It also has a `reg` property, which contains the chips **SPI** address. Next it has a couple of properties prefixed with `spi-`, which can be used for setting up the SPI bus correctly (e.g. maximal bus frequency). The details are not important here.

The next property `vio-supply` describes, that the chip is powered by a voltage regulator labeled `vio` (similar to the CPU power supply information). This information is important for the operating system, so that it can disable regulators when they are not needed (and thus save energy).

The next property, `ti,power-gpio` describes, that the chip is also connected to a GPIO pin provided by the **SoC**. That GPIO can be used for enabling the **WLAN** chip. The two properties following also describe a GPIO connection, but since this GPIO is used as interrupt source its described as such. Last but not least there are two properties, which reference the proper GPIO pin setup for the pin controller.

Listing 2.2: Simplified and Merged SPI4 Excerpt from Nokia N900 Device Tree Source File

```
1 &ocp {
2     mcspi4: spi@480ba000 {
3         compatible = "ti,omap2-mcspi";
4         reg = <0x480ba000 0x100>;
5         interrupt-parent = <&intc>;
6         interrupts = <48>;
7         dmas = <&sdma 70>, <&sdma 71>;
8         dma-names = "tx0", "rx0";
9         pinctrl-names = "default";
10        pinctrl-0 = <&mcspi4_pins>;
11
12        #address-cells = <1>;
13        #size-cells = <0>;
14
15        wl1251@0 {
16            compatible = "ti,wl1251";
17            reg = <0>;
18
19            spi-max-frequency = <48000000>;
20            spi-cpol;
21            spi-cpha;
22
23            vio-supply = <&vio>;
24
25            ti,power-gpio = <&gpio3 23 GPIO_ACTIVE_HIGH>; /* 87 */
26
27            interrupt-parent = <&gpio2>;
28            interrupts = <10 IRQ_TYPE_NONE>; /* gpio line 42 */
29
30            pinctrl-names = "default";
31            pinctrl-0 = <&wl1251_pins>;
32        };
33    };
34 };
```

So the device tree describes quite detailed how different hardware components are connected to each other and how they must be configured for proper interaction. This thesis will depend on this data for properly



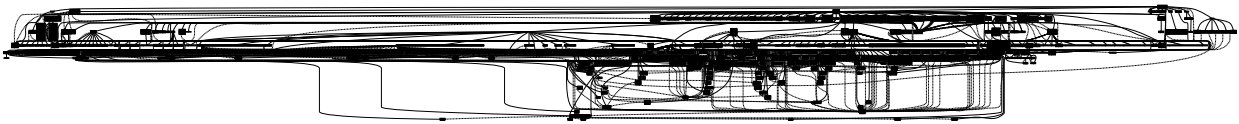


Figure 2.1.: N900 DT graph

describing the hardware by extending the [DT](#)-compiler to support output for [Graphviz](#) files, which can be turned into proper graphs of the hardware resulting in visualized interconnections.

For converting the [DT](#) source files into visual graphs the device tree compiler, which is also shipped with the Linux kernel's source code has been slightly extended, so that it generates a shared object, that can be used with other software.

Based upon this library a small Python proof-of-concept application has been written, which parses the [DT](#) file and creates a [Graphviz](#) node for each node found in the [DT](#). Additionally links are added for each parent-child connection in the [DT](#) file as well as for each phandle<sup>4</sup> reference.

As visible in [Figure 2.1](#) the graph contains far too many nodes to be usable, though. For this reason a couple of parameters have been introduced to reduce the nodes appearing in the graph. This can be used to have a look at specific details of the platform's interconnections.

For example [Figure 2.2](#) is a [DT](#) graph visualization starting at the OMAP's system bus with a maximum depth of one node, with disabled rendering of any properties attached to the nodes. The result is a nice overview of all [IP-Cores](#) available on the OMAP3430 [SoC](#). As shown in the figure there are almost 80 [IP-Cores](#) available in the [SoC](#) with some of them missing in the graph, since they do not yet have a [DT](#) binding (e.g. the [Image Signal Processor \(ISP\)](#) used for connecting cameras).

Describing the whole architecture would be out of scope for this thesis, but to visualise the interconnection principles a more detailed graph for the second [I2C](#) module is shown in [Figure 2.3](#). In opposit to [Figure 2.2](#) the property rendering has not been disabled, so that hardware details become visible. Shown in [Figure 2.3](#) is the address space of the second [I2C](#) module (0x48072000 - 0x48072080), as well as the interrupt port it is connected to (0x39 → 57). The interrupt controller is inherited from its parent node (the [OCP](#) bus), which is not part of the figure. The next entry ("dma") is actually a merged property from the [DT](#) file to ease reading. The properties define a [DMA](#) channel for receiving and sending data respectively. In opposit to the interrupt property the [DMA](#) controller is explicitly referenced via its label. Next the pin controller configuration is referenced, which is supposed to configure the pins for usage with the [I2C](#) module. Last but not least the [I2C](#) module is configured to use a bus speed of 100 KHz.

The child nodes of the controller are no longer part of the [SoC](#), but standalone chips connected to the OMAP via its second [I2C](#) port. Each child node provides a *reg* property, which describes the chip's [I2C](#) address. Also provided by each node is a *compatible* property, which identifies the chip. Other common properties are [GPIO](#) specifiers, which, like the [DMA](#) channel description, reference a controller and provide an additional identifier for the pin number. The additional integer describes if the [GPIO](#) is used with an active high or an active low signal. The properties and their values are partly defined directly in the ePAPR standard, partly generic extensions of the ePAPR standard and sometimes device specific. Normally device specific properties are supposed to be prefixed by the manufacturer code, which is also used in the *compatible* string (like e.g. the "ti,resistor-sense" property in the bq24150a node).

<sup>4</sup>phandle is the [DT](#) term for the references seen in [Listing 2.1](#) and [Listing 2.2](#)

## 2. Hardware Architecture

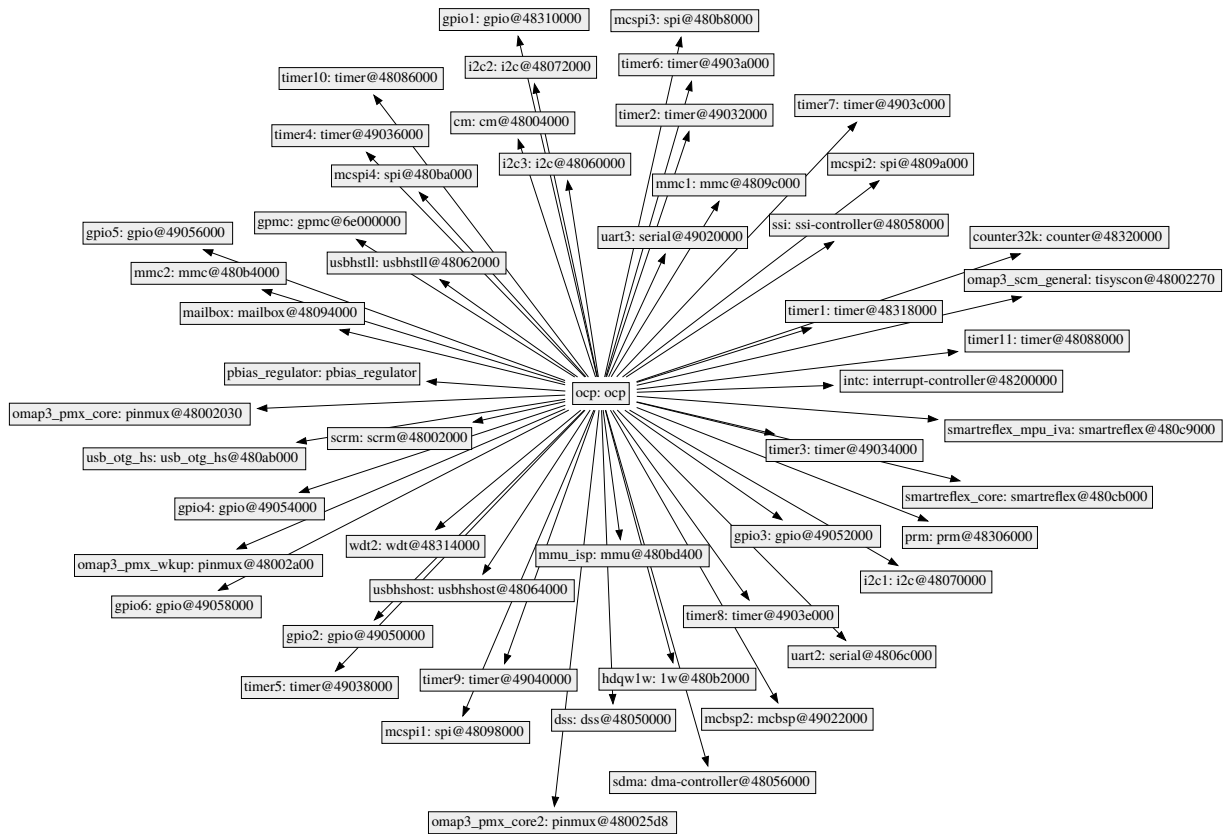


Figure 2.2.: DT graph of OMAP3430's system bus

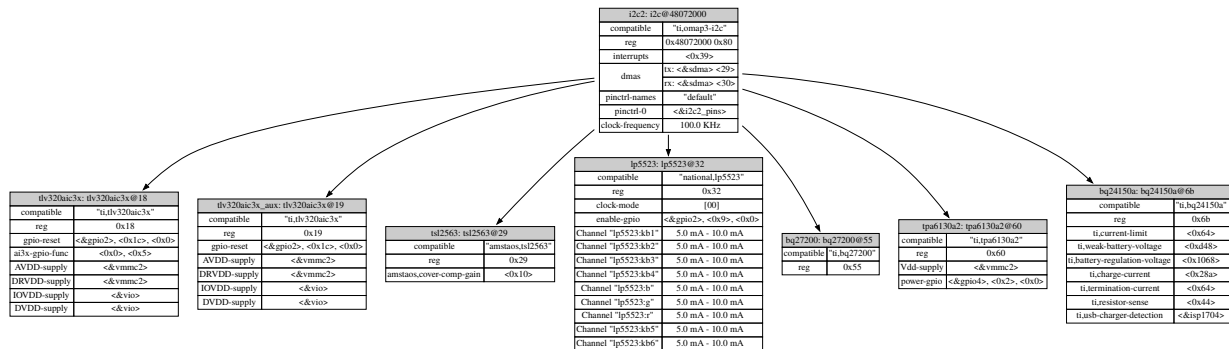


Figure 2.3.: DT graph Nokia N900's second I2C port

While none of the components connected to the second I2C bus is strictly needed (a description for each component can be seen in the list below), some other externally connected components are required for booting.

- tlv320aic3x: audio codec from [Texas Instruments](#)
- tsl2563: light sensor from AMS TAOS
- lp5523: multi-channel LED controller used for the keyboards backlight and the phone's RGB led
- bq27200: battery fuel gauge
- tpa6130a2: headphone amplifier
- bq24150a: Li-ion battery charger

For example to get data on the built-in [Liquid Crystal Display \(LCD\)](#), a couple of different subsystem must be initialized before. First of all the OMAP processor contains an [IP-Core](#), which is named [Display Subsystem \(DSS\)](#). This module acquires its frames from the system's memory using [DMA](#), optionally adds up another image (e.g. the mouse cursor's sprite) and finally outputs the frame via [Texas Instruments's Serial Display Interface \(SDI\)](#), [MIPI's Display Serial Interface \(DSI\)](#) or its TV out encoder.

In addition to the [DSS](#), obviously the [LCD](#) itself must be configured correctly. Since it is connected to the first [SPI](#) port, the [SPI](#) subsystem must also be configured. In addition the display requires a reset [GPIO](#), resulting in a dependency on the [GPIO](#) subsystem. It should be noted, that this setup only supports simple frame buffers, but no 2D or 3D acceleration. Accelerated graphics require support for the [PowerVR SGX IP-Core](#).

For developing a more simple interface to the system is needed, so that early errors can be analysed. Apart from that using the [LCD](#) means, that it may not be possible to read the full error message (most stack traces do not fit on a 800x480 screen) and the messages cannot be copied for later analysis. For Linux kernel development usually a [UART](#) interface is used, which usually can be initialized quite early in the boot process or is already pre-configured by the bootloader[21, p. 365]. Apart from that accessing it is quite simple compared to other devices.

### 3. Serial Access

This chapter will describe how access to one of the **OMAP's UART** ports, also named serial interface will be gained. Once the serial controller is initialized, which can be very early depending on the kernel configuration, the kernel's main console can send data to it. Thus the debug messages of all drivers will be routed via the **UART** interface <sup>1</sup>. Other options for debugging are more fragile, initialized too late or just not as convenient:

One of the most obvious solutions is usage of the built-in **LCD**. This solution has a couple of disadvantages, though. Having the debug messages displayed means, that only the last few lines are visible and one has to type out the message. There are also disadvantages from the drivers side of view. For accessing the display several subsystems must be configured correctly: As mentioned in ??, the display controller is connected via **SPI** and uses a **GPIO**, so these subsystems have to be initialized. Additionally the **OMAP's DSS** controller must be configured, which makes use of **DMA**. On a typically configured kernel the display is thus one of the last initialized components. Apart from the increased amount of subsystems, that have to be configured, several of the involved controller are much more complex than the **UART** controller, so there is an increased chance of something going wrong before the display is configured as debug output.

A second solution for accessing debug messages is the **Universal Serial Bus (USB)** port. It has the ad-

---

<sup>1</sup>The kernel's message output framework ensures, that a message is fully sent before it starts sending the next message.

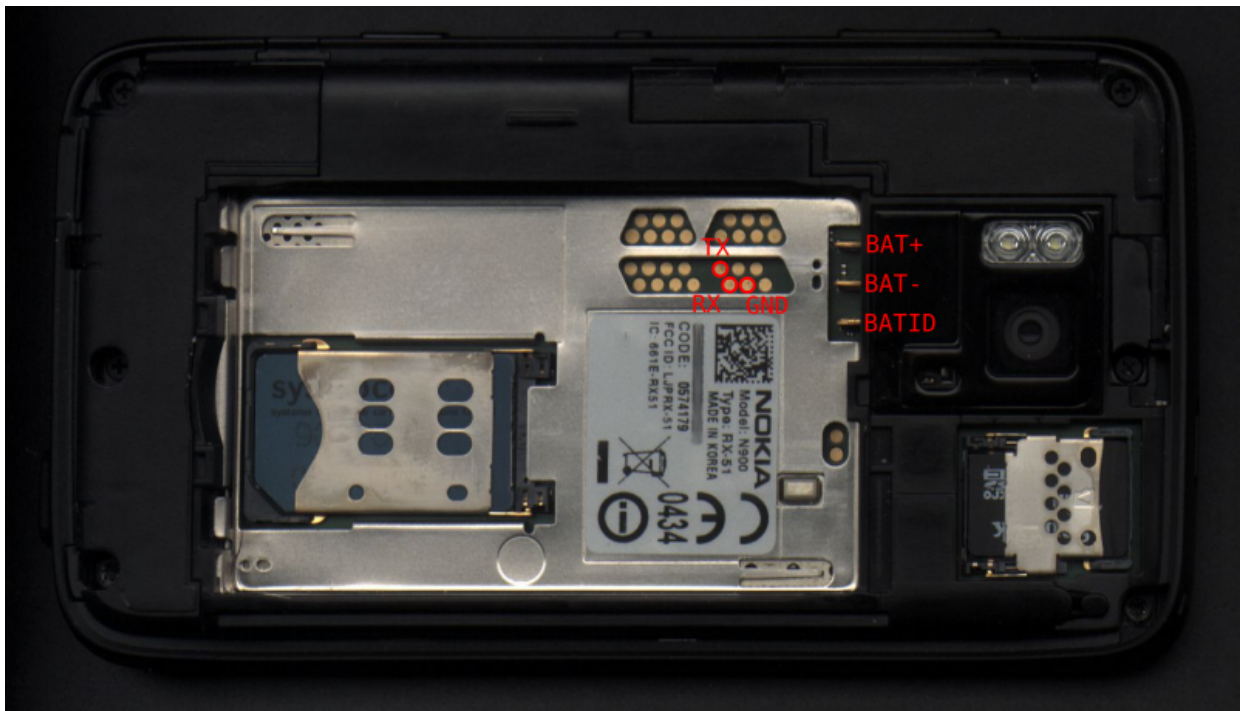


Figure 3.1.: Coverless N900's back with labeled serial test pads

vantage of better throughput, but like the graphic stack its dependent on multiple other subsystems and its initialization is error prone.

Another variant is using a working userland and access the kernel messages from there. This is the easiest method, but only works if the userland can be started. This limits its usability for kernel development, since the kernel may crash before the userland starts.

Fortunately there are pad connectors under the N900's battery compartment. These have been analyzed by the Maemo community [22] and some pads found to be connected to the SoC's third UART port. The test pads needed for the serial access are marked in Figure 3.1. Two problems must be solved before the UART port can be used, though:

1. The pads are neither a standard RS232 connector using 12V, nor using a 3.3 or 5 Volt based voltage level. Thus it is not possible to simply use a USB-RS232 or USB-UART adapter, which are commonly found at electronic distributors<sup>2</sup>. This can be solved by either using a more expensive and less common USB-UART adapter supporting a 1.8 Volt IO voltage or by using a 5V based USB-UART adapter together with a logic level converter between the test pads and a USB-UART adapter.
2. If cables are connected without modifying the case, it is not possible to insert a battery. Nokia's developers had an adapter to access the pads, but it has never been publicly accessible. If the phone is used only for development one could simply solder wires directly to the test pads and the battery connectors. Otherwise Pogo-Pins (also known as spring probe pins, see Figure 3.2) can be used together with a small mounting, so that they probe the correct pads. The fact, that the pads are below the battery compartment can be used as an advantage here, since the battery compartment can hold the self-built adapter instead of a battery.

For solving both problems a simple debug adapter will be designed in the following paragraphs. This adapter uses Pogo-Pins to tap the UART test pads below the battery compartment and for tapping the battery pins of the compartment. To tap the correct pads a dummy battery will be built, which has holes to fit the Pogo-Pins. Then a small circuit will be created, which does a 5V to 1.8V voltage level conversion for the N900's UART port and contains a USB-UART chip, so that a simple USB cable is enough to access the phone's UART interface.

Additionally two optional features will be added to the circuit making it useful for more advanced use cases in the future:

1. A power meter is put between the battery and the phone, so that the phone's energy usage can be measured. This is useful to check the effectiveness of different power saving methods.
2. A kill switch is added between the battery and the phone making it possible to interrupt the phone's power supply. This is needed to reboot the phone in case of a hanging kernel and allows to use the adapter for automatic regression testing once everything works as expected.

Designing the adapter requires information about the exact positions of the UART pads first. This can be

---

<sup>2</sup>Using these adapters directly will most likely destroy the complete SoC or at least the OMAP's UART port



Figure 3.2.: Pogo-Pin profile

### 3. Serial Access

done manually (e.g. by measuring the distances using a vernier scale), but for this thesis a simpler and precise method has been used instead: Scanning the phone in a flatbed scanner returns a pixel based image of the battery compartment, which can be used as a basis for vectorization.

The vectorization task can be seen in [Figure 3.3](#). The red and blue lines drawn above the scanned image are vector information, while the underlying image is a pixel based. Once everything of interest has been vectorized, the pixel based image can be removed from the file resulting in a [Scalable Vector Graphics \(SVG\)](#) file, which can be further converted to other vector formats understood by [Computerized Numerical Control \(CNC\)](#) machines.

Before the vector information can be used with a [CNC-machines](#) it must be scaled accordingly, though. Since [CNC-machines](#) ignore the thickness of the vectors, the software used for scaling should ignore it, too. In Inkscape this has to be configured by selecting enabling [Preferences](#)→[Tools](#)→[Geometric bounding box](#). Afterwards one can measure one side of the battery slot and scale the whole object by locking width and height together and providing the correct information for one of the sides.

Next a simple test production should be done to verify, that everything is correctly sized and positioned. This should be done using transparent material, so that one can see if something stucked into the wholes for the [UART](#) pads hits them correctly.

Once the physical information has been acquired the circuit design can be started. Since the adapters main intention is accessing the phone's serial port and most modern computers/notebooks no longer provide any

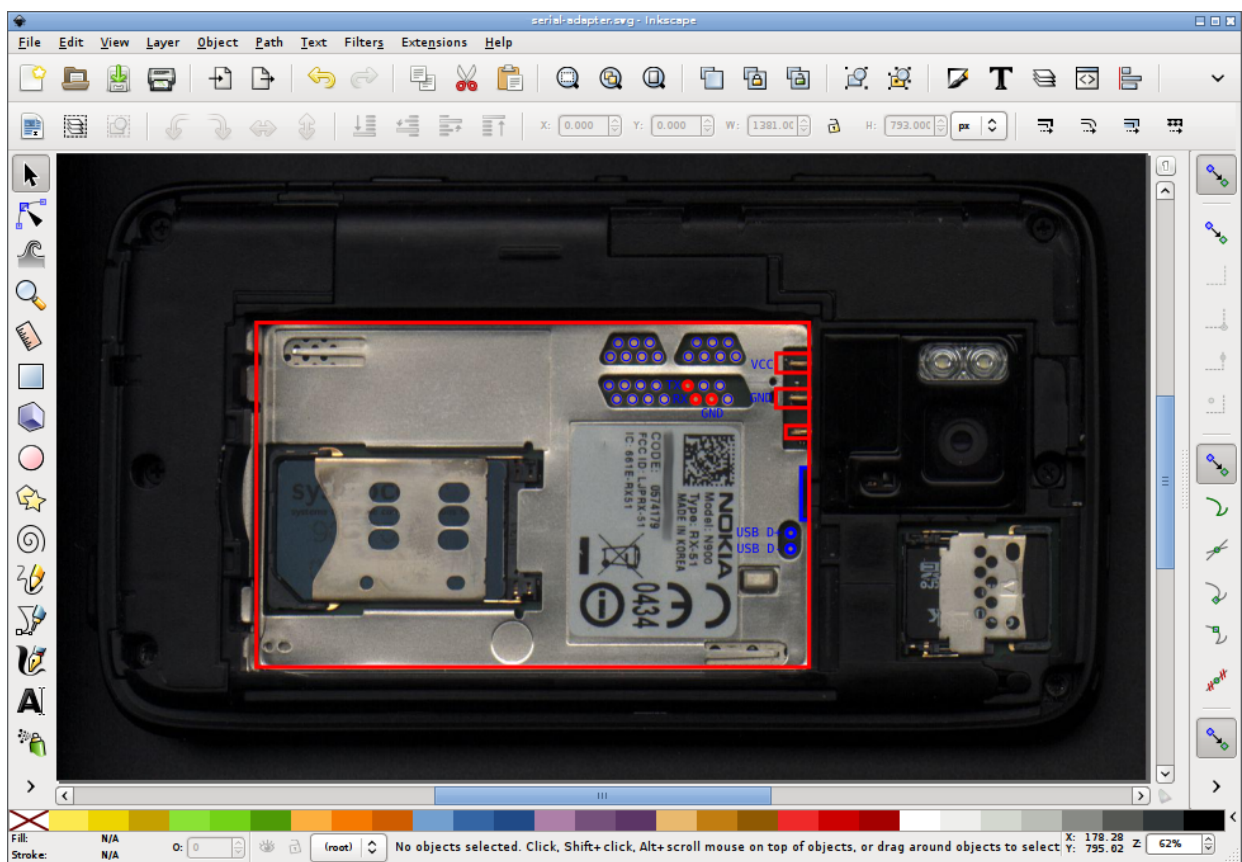


Figure 3.3.: Battery adapter design in Inkscape

RS232 port, a USB to UART chip will be used as base. For additional voltage and current measuring an additional port with the custom logic will be needed. This port should support I2C or SPI instead of the UART protocol making it easy to acquire sensors for the power measurements.

For this thesis FTDI's FT2232H and Cypress's CY7C65215 USB to dual UART solutions have been evaluated. Both chips provide support for two serial interfaces, which can be configured as I2C, SPI or UART independently. Since the CY7C65215 has an integrated oscillator making it a bit easier to integrate and is also a bit cheaper than the FT2232H it has been chosen here.

Since the N900's UART pins must be driven at 1.8V and USB provides only 5V, a voltage controller is needed. The power is only needed as reference and thus a very small voltage controller is enough. For this thesis an Texas Instruments TPS76918 has been chosen, since it was available from local distributors and can be soldered surface mounted making the case design easier.

For power measuring Texas Instruments's INA219 has been chosen, since it was the only fully integrated solution available from local distributors. This introduces another problem: The UART pins must be driven at 1.8V, but the INA219 must be driven between 3V and 5V.

This has been solved by adding a transistor to the RX and TX lines between the phone and the CY7C65215, which does the logic level voltage translation. For this task a Fairchild Semiconductor BSS138 has been chosen, since it has been shown to be suitable for this purpose in SparkFun's Logic Level Converter board.

Last but not least another transistor has been added between the phone's battery connector and the real battery, so that the power supply can be cut on demand. The complete schematic can be seen in Figure 3.4.

In the next step all components must be connected to their footprints, since the schematic only contains information about the abstract components. To protect the PCB against physical damage and to make the whole adapter more portable the PCB is intended to be placed inside of the adapter. This requires cutouts in the adapter's case for every part soldered on the PCB. To avoid doing this work on both sides of the PCB, only Surface Mounted Device (SMD) footprints have been selected for the PCB in this thesis.

Once all components have their footprints assigned the PCB layout can be created. This task requires some additional obstacles compared to simple PCBs, since the drilled holes for the underlying debug pads must be positioned exactly. This has been solved by starting to draw the PCB's edges first using the size informations from the vectorized data created before. Then in the next step the drilled holes can be positioned using the vectorized data.

Before continuing to place and connect all components in the PCB-software, one can validate the correct position of the drilling holes by exporting the PCB data into an intermediate format, which is supported by the vector-software (e.g. PostScript). Then the PCB data can be put as overlay on top of the vectorized data. If the drilling holes do not match the vector information they should be relocated appropriately.

Once all PCB holes sizes and positions match the ones from the initial vector file, all components have been placed and the traces have been routed, the PCB is ready for manufacture.

There are a couple of methods available for creating PCBs prototypes. Normally a PCB consists of a copper layer on top of support material. The big industrial machines used for PCB manufacturing use acid to etch some areas of the copper layer leaving the separated traces. Alternatively a CNC-mill can be used to remove the copper layer as needed. This method leaves less hazardous waste (no used acids), but is not interesting for industry-grade manufacturing, since the process requires more time (acid works on

### 3. Serial Access

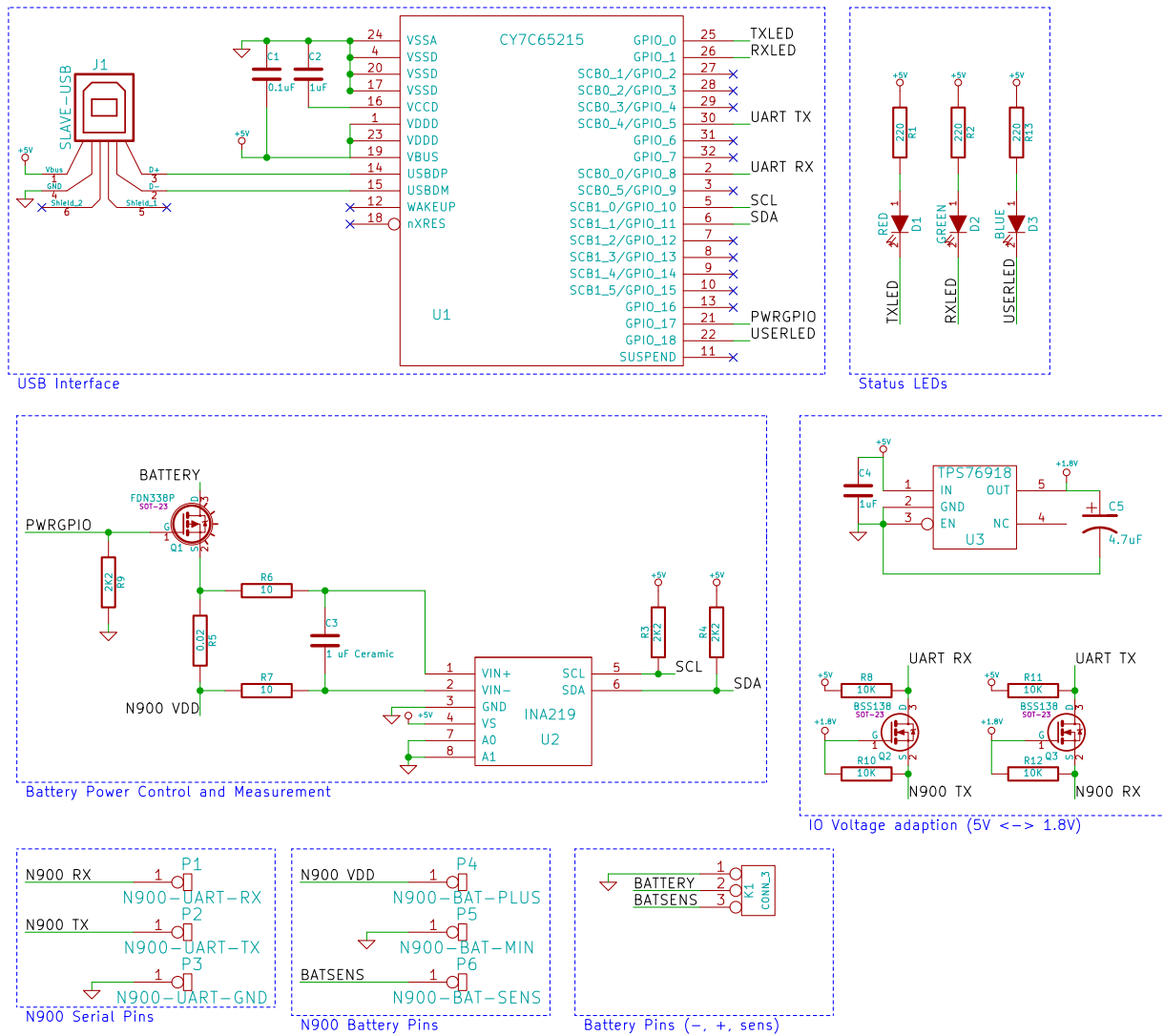


Figure 3.4.: Serial adapter schematic (designed with KiCad)



the whole board at the same time) and is pricier (milling heads must be replaced regularly). Nevertheless isolation milling is a fast way to create a low quantity of prototypes.

**Method I: CNC-based** For this task the output usually generated by PCB-software (Gerber files), must be post-processed to generate instructions for the CNC-mill (which usually requires G-Code). One of these software files is visible in [Figure 3.5](#). The green areas are read from the Gerber file and are supposed to become isolated copper areas. The isolation milling software solves the problem by generating paths for the CNC-mill around the green areas (red lines). The path has a little distance from the green areas to compensate the milling tool's diameter. As a result the tool's diameter must be smaller than the area, that should be removed. For example the approximately 0.4mm tool used in [Figure 3.5](#) is too big to mill between the pads intended to be used for the micro-controller.

**Method II: Acid-based** After some experimenting the idea of using isolation milling to create a prototype for the adapters PCB has been given up, since the required milling tools were not available easily. Instead the etching method has been tried by exposing the PCB to ultra-violet light with a protection mask on top of the areas, which should not be etched. Next the PCB is put into sodium hydroxide to remove the light-sensitive coating from the areas exposed to the ultra-violet light. Last but not least the PCBs were put into Iron(III) chloride to etch away the copper layer. Unfortunately the results were not good enough to be usable as prototype.

**Method III: Professional Manufacturer** At the end the PCB files have been sent to a professional PCB manufacturer without prior prototype testing. The resulting PCB can be seen in the left of [Figure 3.6](#).

In the next step the PCB had to be populated with the SMD parts. This task can be done either by using a thin soldering iron or by applying some soldering paste on the pads, putting the part on top and heat up the paste using hot air or an oven. Soldering the QFN32 micro-controller using a soldering iron is quite hard, so soldering paste has been used instead. For properly applying the paste on the pads (correct amount and position), a simple stencil has been created using a Thunderlaser Mars 120 laser-cutter.

Professional stencil are usually created using a thin steel or aluminium sheet. Unfortunately the Thunderlaser Mars 120 cannot cut metal, so instead a sheet of transparency usually used with overhead projectors has been tried as suggested by different Blog articles in the web [3]. While the resulting stencil was good enough for the bigger SMD components, the QFN32 pads became too big making the stencil unusable.

Instead of the transparency a few other materials have been tested after some unsuccessful results. At the end simple crepe tape produced very good results. A PCB prepared with the crepe tape stencil can be seen on the right in [Figure 3.6](#). In opposit to standard stencils it has the advantage of being sticky, so its very easy to distribute the solder paste without accidently moving the stencil. Instead it becomes a little harder to remove the stencil without ruining the solder mask on the PCB. This turned out unproblematic if only small areas of the PCB are prepared. A second disadvantage of the crepe tape is its rough surface resulting in wasted soldering paste.

Once the PCB has been fully assembled, it must be prepared for usage as battery replacement in the N900. For this task a case out of acrylic glass has been created. [Figure 3.7](#) depicts the vector data, which will be used with the CNC machine. Depending on the layer different colors are enabled for being cut resulting in the setup seen in [Figure 3.8](#). All in all three different layers are needed between the PCB and the phone's battery compartment and five additional layers are required on top of the PCB.

### 3. Serial Access

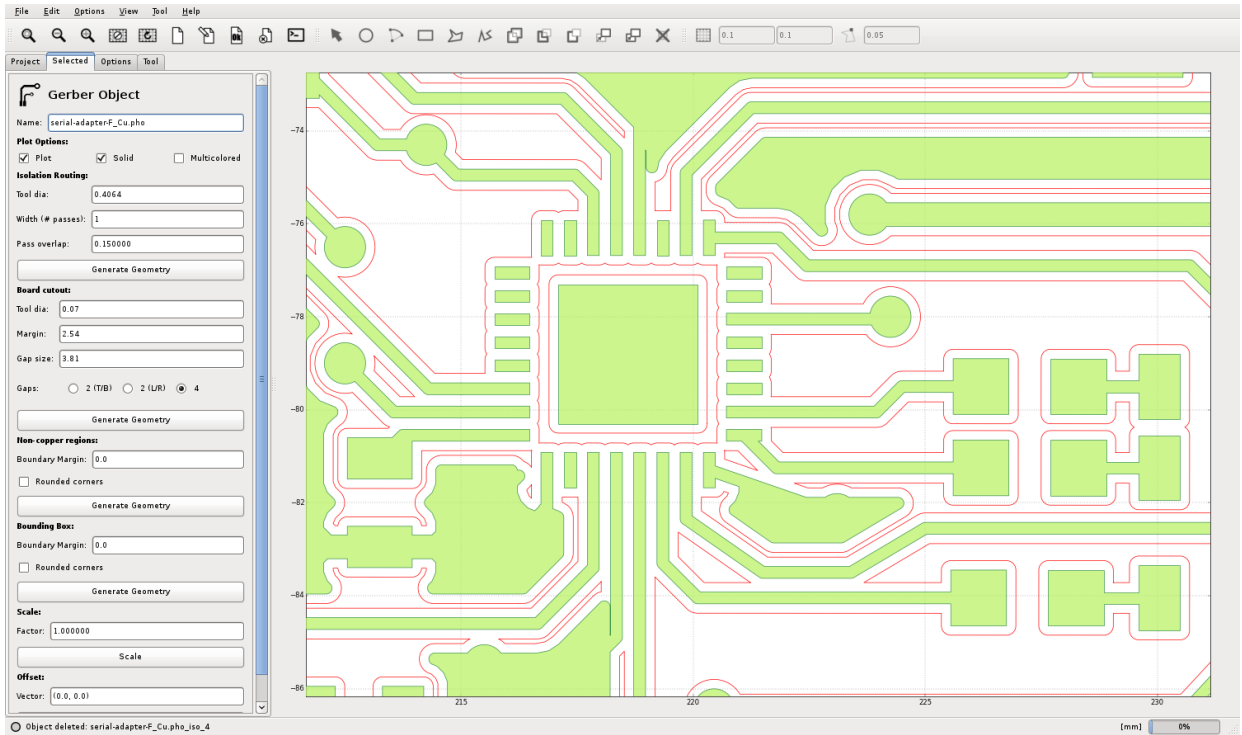


Figure 3.5.: Generating G-Code for Gerber-files using FlatCAM

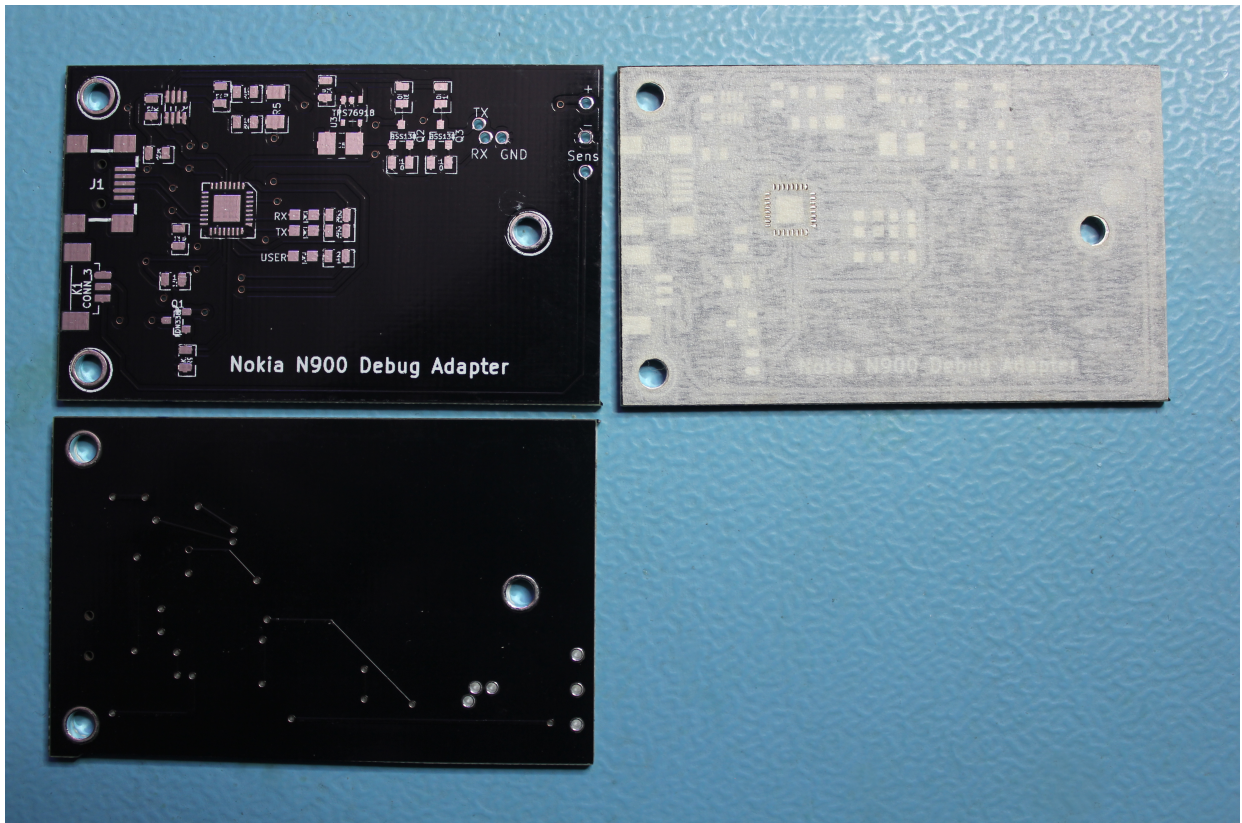


Figure 3.6.: left: Front and back of the adapter's PCB; right: PCB with crepe-tape-stencil

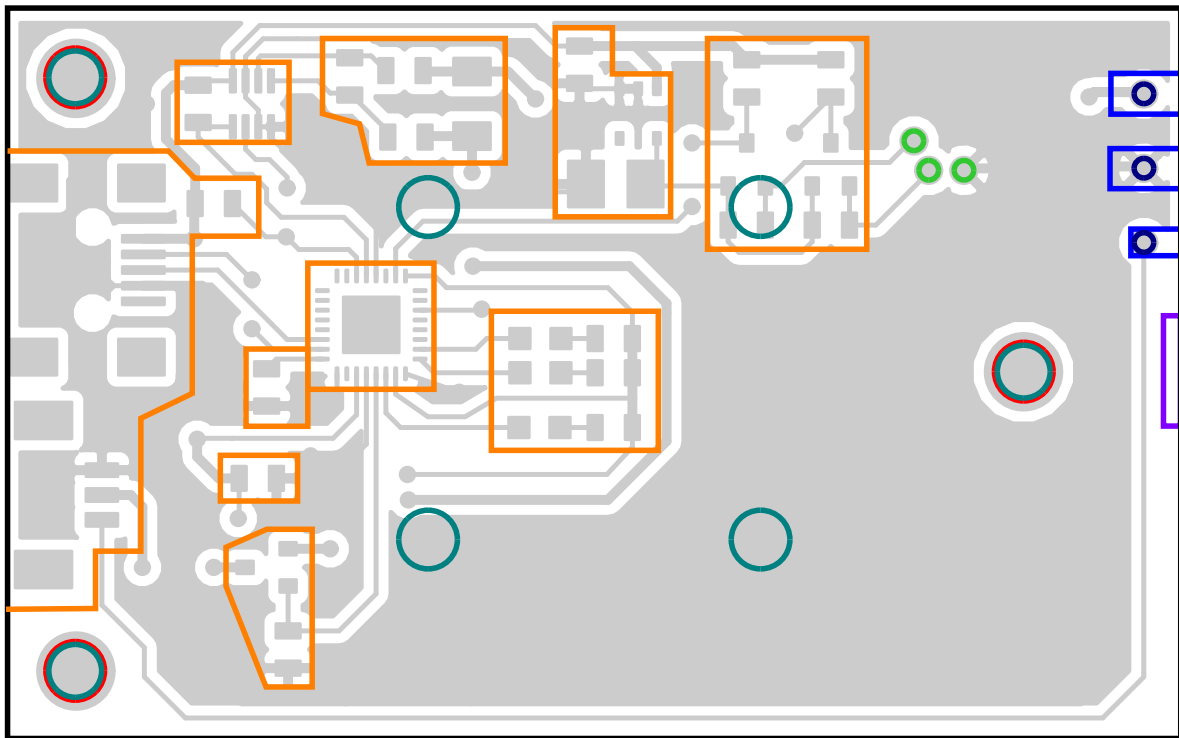


Figure 3.7.: Colorized vector information for the laser cutter

All layers are cut with the black, outer border, but only the lowest layer enables the blue rectangular vectors (■) for the phone's battery connectors. Also enabled in the lowest layer are the green-blue circles (●), which are inside of the red circles. Those are a little bit smaller than the screws, which will be used to hold everything together. This will result in the screws cutting their own screw thread. Last but not least the green small circles (●) for the serial pins are enabled. Using a little bit of the acrylic glass' offcuts it has been tested, that the bottom layer should be 3mm thick.

The next layer enables the purple box (■), so that the phone's nose intended to hold the battery tightly can do the same with our adapter. Additionally the dark-blue circles (●) are enabled instead of the blue rectangular boxes, since the phone's battery pins are less than 3mm in height and the round holes provide the **Pogo-Pins** stability. Like the previous layer, this layer contains the holes for the serial **Pogo-Pins** (●) and still enables the smaller screw holes (●). Like the first layer this one should be about 3mm in height.

The third layer must disable the purple box (■) again, so that the adapter can be clamped into the battery compartment. Like the previous layer all six holes for the **Pogo-Pins** must be enabled (●, ●), too. To simplify the construction step the third layer should no longer use the small screw holes though. Instead the bigger, red holes (●) should be enabled. The height of this layer should be at least 3mm (to provide some distance between the **PCB** and the phone), but can be bigger. The prototype built in this thesis 6mm have been chosen.

The next layer is the **PCB**, which can simply be put on top of the acrylic glass. If the **Pogo-Pins** have not yet been soldered to the **PCB**, it should be done now. For this task it helps to put a 2mm layer below the other layer while keeping the battery holes uncovered. Inserting the **Pogo-Pins** into the prepared holes in the **PCB** and the adapter's lower acrylic glass layer should result in the pins sticking 2mm out of the adapter.

### 3. Serial Access

Once this has been ensured the pins can be soldered to the PCB. Next the additional layer at the bottom should be increased to 5-6mm and the same procedure should be done for the serial pad's Pogo-Pins, so that they stick out 5-6mm. Once this has been done the PCB is very fragile, so all further steps should be done with care.

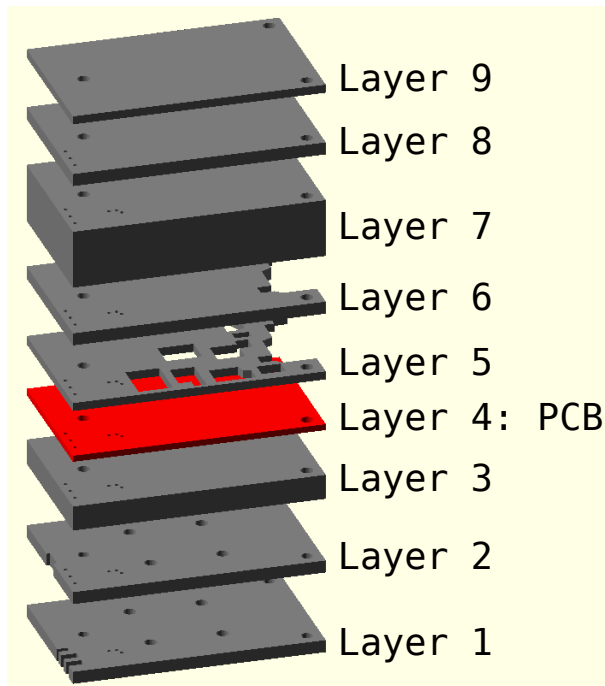


Figure 3.8.: 3D-rendering of the layers

The layer following right after the PCB must enable a cutout for the parts on top of the PCB by enabling the orange vectors (orange square). Apart from that the red screw holes (red square) and the Pogo-Pin holes (green square, blue square) must be enabled. This layer should have a height of 5mm, which is a bit more than the height of the Mini-USB port soldered to the PCB.

The next few layers are basically identical to the previous layer, but without the orange PCB-parts cutout. A couple of them should be stacked to reach the end of the serial Pogo-Pins. At this point the next few layers should not cut the green (green square) holes, so that the Pogo-Pins are protected. The same procedure is done for the battery pins, which should be roughly 3-4mm longer. Once the final layer has been put on top the red screw holes (red square) can be used with 3mm machine screws.

Since the Pogo-Pins must stick out at the bottom of the adapter and they are quite fragile, transporting the adapter as-is, might be a problem, though. For this reason the lowest two layers should contain the four green-blue holes (green-blue square) in the middle. They can be used to hold a simple protection cap.

The protection cap can be made by cutting another layer, which has all six Pogo-Pin holes (green square, blue square) and the green-blue holes (green-blue square) in the middle of the adapter, though slightly reduced in their size. This layer should be at least 6mm long, so that the protruding Pogo-Pins are completely covered. Next four 10mm long M3 screws can be inserted into the protection cap layer. To prevent the protection cap from falling of the adapter an elastic band can be used.

Since the described adapter does not take care of holding the battery, a battery compartment must be build, too. For this task a cheap (apparently broken) Nokia phone using the same battery type has been acquired. Next the phone has been disassembled to get its PCB, which in turn has been further processed with hot air to remove all parts except the battery pins.

At the next step its important to verify, that there are no electrical shorts between any of the battery pins (in that case another hot air rework is needed to remove remaining solder from the PCB's pads).

Then a frame was designed using the same technique as for the actual adapter. All PCB parts, which are not covered by the frame have been cut off and some wholes were drilled through the more or less unused PCB as needed. Last but not least the PCB is put between the frame layers and a wire is soldered to each of its battery pins. Once a plug matching the debug adapter's power socket has been crimped to the wires the battery compartment is ready to use.

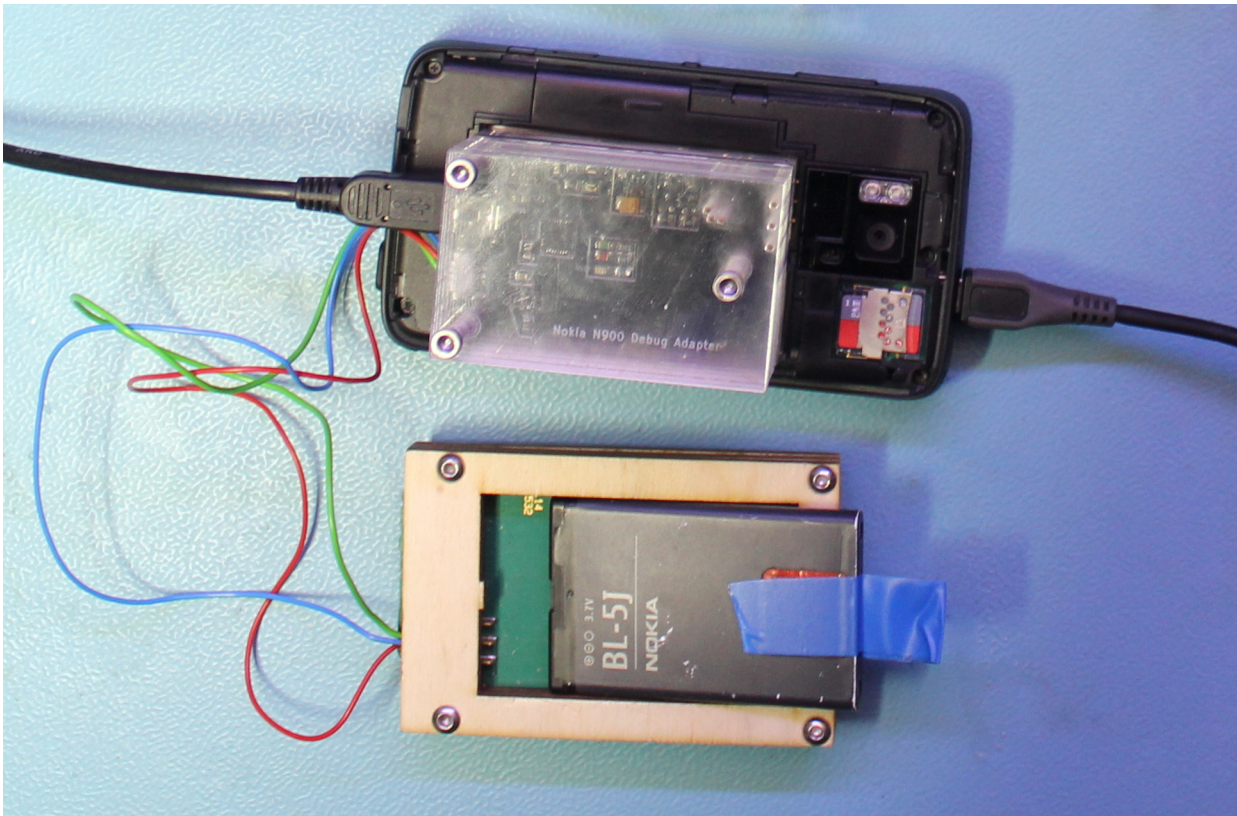


Figure 3.9.: N900 with attached debug adapter

With both, the adapter and its battery compartment it is possible to exchange data with the N900's serial port. [Figure 3.9](#) shows the complete setup. If the PC's serial software is configured to the default serial settings used by the N900, which are 115200 baud and 8 data bits, no parity bit, 1 stop bit (8-N-1 format), it is possible to receive [NOLO's](#) debug output as visible in [Listing 3.1](#)

#### Listing 3.1: Nokia N900 Boot Log

```
NOLO X-Loader (v1.4.14, Jun  3 2010)
Secondary image size 109384
Booting secondary
[ 0.002] Nokia OMAP Loader v1.4.14 (Jun  3 2010) running on Nokia N900 F5 (RX-51)
[ 0.014] I2C v3.12
[ 0.033] System DMA v4.0
[ 0.036] OneNAND device ID 0040, version ID 0121 (256 MiB, 66 MHz)
[ 0.070]   OneNAND blocks unlocked in 28010 us
[ 0.075]   Flash id: ec4021
[ 0.097] Partition table successfully read
[ 0.105] TWL4030 PWR ISR: 44
[ 0.108] Reset reason: usb
[ 0.111] McSPI v2.1
[ 0.114] LP5523 found at I2C bus 2, address 0x32
[ 0.125] SMB138C: Not loading driver (version reg. 0x4b)
[ 0.131] BQ24150 (rev. 3) found on I2C bus 1, address 0x6b
[ 0.137] SSI version 1.0
[ 0.151] Battery voltage 4.020 V, BSI: 0
[ 0.161] Disabling charging (no battery present)
[ 0.170] Initializing LCD panel
[ 0.173]   Detecting LCD panel moscow
[ 0.177]   Panel ID: 108b77
[ 0.180]   Detected LCD panel: acx565akm
[ 0.184] DISPC: version 3.0
[ 0.189]   LCD pixel clock 24000 kHz
[ 0.220]   Logo drawn in 5 ms (11700 kB/s)
```

### 3. Serial Access

```
[ 0.350] Über-cool backlight fade-in took 9 ms
[ 0.358] Initializing USB
[ 0.366] USB host detected
[ 0.369] Entering USB loop
[ 0.420] USB suspend signaling detected
[ 0.544] USB reset received
[ 0.655] SETUP: RD STD DEVICE GET_DESCRIPTOR DEVICE (value 00, index 0000, length
64 bytes)
[ 0.669] USB reset received
[ 0.831] SETUP: WR STD DEVICE SET_ADDRESS value 0015 index 0000 length 0000
[ 0.855] SETUP: RD STD DEVICE GET_DESCRIPTOR DEVICE (value 00, index 0000, length
18 bytes)
[ 0.865] SETUP: RD STD DEVICE GET_DESCRIPTOR CONFIGURATION (value 00, index 0000,
length 9 bytes)
[ 0.875] SETUP: RD STD DEVICE GET_DESCRIPTOR CONFIGURATION (value 00, index 0000,
length 94 bytes)
[ 0.886] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 00, index 0000, length
255 bytes)
[ 0.895] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 02, index 0409, length
255 bytes)
[ 0.905] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 01, index 0409, length
255 bytes)
[ 0.915] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 05, index 0409, length
255 bytes)
[ 0.925] SETUP: WR STD DEVICE SET_CONFIGURATION value 0001 index 0000 length 0000
[ 0.934] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 03, index 0409, length
255 bytes)
[ 0.943] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 04, index 0409, length
255 bytes)
[ 0.953] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 04, index 0409, length
255 bytes)
[ 0.963] SETUP: WR STD INTERFACE SET_INTERFACE value 0000 index 0002 length 0000
[ 0.972] SETUP: RD STD DEVICE GET_DESCRIPTOR STRING (value 04, index 0409, length
255 bytes)
[ 2.353] Timeout waiting for flashing commands
[ 2.363] Loading kernel image info
Loading kernel (1961 kB)... done in 71 ms (27434 kB/s)
[ 2.443] Loading initfs image info
[ 2.447] Total bootup time 2472 ms
[ 2.451] Serial console disabled
Uncompressing Linux.....
..... done, booting the kernel.
```

Also visible in [Listing 3.1](#) is a hardware problem of the adapter built for this thesis. Apparently the phone's bootloader (and the original Maemo operating system) has problems to read out the *BSI* value, which is the third battery pin (used for identifying the battery). This results in no charging being performed. Since the *BSI* value is currently ignored by the charging driver, which is part of the mainline kernel, this is not a problem there.

Charging the battery reverses the voltage flow though, resulting in the adapter's power control switch transistor being used in reverse direction. This results in the transistor being degraded to a diode, reducing the voltage going into the battery by approximately 0.7V. This results in a reboot-loop once the battery is below a threshold capable of supplying the phone. This has been fixed by replacing the power control switch transistor with a direct connection between the phone and the battery effectively removing the adapter's extra functionality. In a future revision of the adapter the transistor should be replaced by a relay.

With access to debug output from the phone it is possible to start working on the kernel without relying on the lower stack being functional. In next chapter we will focus on kernel development for the N900. First starting with an introduction, which describes the current state.

## 4. Linux Kernel

At the base of each [Linux distribution](#) is the Linux Kernel providing the operating system's core. The Linux kernel takes care of abstracting the underlying hardware and thus is one of the most important components for the operating system's smartphone support.

Since most smartphones come with Android today, there actually is already a Linux kernel tree, which contains the device drivers required for a phone. Usually the kernel trees used by the vendor only support one specific phone, so that multiple kernel trees must be taken care of. Most major [Linux distributions](#) do not support multiple kernel sources, though. Instead they use the *mainline* kernel tree provided by [Linus Torvalds](#) together with a few distribution specific patches. Adding custom changes to the kernel results in increased maintenance efforts. As a result most [Linux distributions](#) only add patches, that have already been accepted in the mainline kernel. Some distributions provide multiple kernel source trees to avoid the additional integration work by simply using the vendor's kernel tree. This has the disadvantage, that the vendor's kernel tree is often unmaintained after a short amount of time and its hard to track security updates.

The problem with vendor kernels can be seen in the bottom graph of [Figure 4.1](#) (solid arrows stand for a full merge of all patches and dashed arrows for a partial merge), which shows how a vendor kernel is usually developed: At some point a mainline kernel release is forked and the vendor adds its customizations. Then the mainline kernel development proceeds. At some point the vendor merges the changes (one

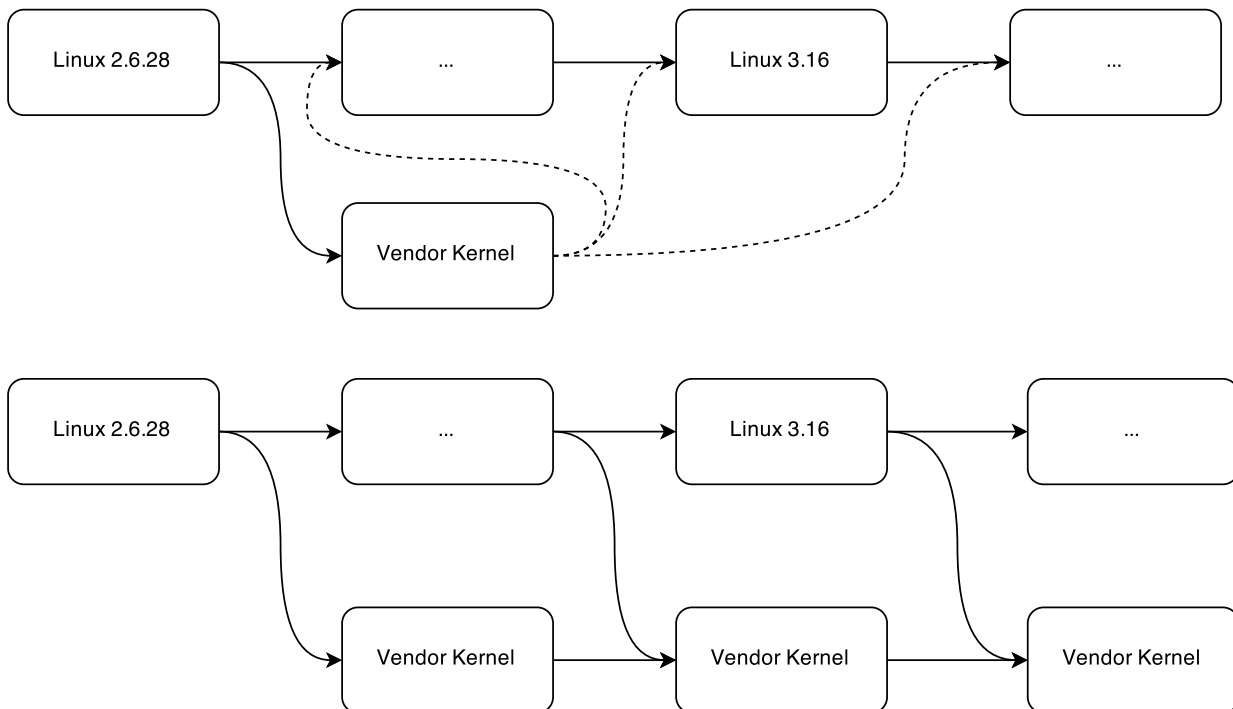


Figure 4.1.: Flow-graph of kernel forks

#### 4. Linux Kernel

mainline kernel release currently has about 10.000 changes [36]) done in the mainline kernel into their own development tree. For Android based vendor kernels its even more complicated, since the process happens two times: There is a kernel fork from Google, which adds the Android patches and then there is another fork done by the vendor, which adds the device specific changes on top.

The kernel developers would prefer to merge the changes into the mainline kernel instead, resulting in less work for all involved parties. For the vendor it means, that the driver is taken into account, when internal kernel [Application Programming Interfaces \(APIs\)](#) change. For users it means, that the community has a chance to keep their device working once the vendor has stopped updating their vendor kernel and for the kernel developers it broadens the pool of devices considered for designing kernel frameworks. The downside of the development style desired by the community are longer product development cycles. Getting full support for a device into the mainline kernel takes multiple months and may take multiple years depending on the amount of missing features. Apart from that it is not guaranteed, that a specific [API](#) developed by the vendor is accepted in the mainline kernel.

The Nokia N900 has been delivered with a Linux 2.6.28 based kernel, that includes many additional device drivers. This kernel tree has never been updated to a newer kernel basis by Nokia, but Nokia's developers send many of those drivers to their respective maintainer in the mainline kernel. A simplified graphical representation of this development style can be seen in the upper graph of [Figure 4.1](#).

To get patches into the mainline kernel it is useful to understand how its development flow works nowadays. As seen in [Table 4.1](#) new kernels have been released by [Linus Torvalds](#) approximately every eight to nine weeks. This is the result of the kernel's development flow used for the last few years as visualized in [Figure 4.2](#). At the beginning of each kernel release [Linus Torvalds](#) pulls from different topic branches, which have their own maintainers. Usually he closes the so called merge window after two weeks by releasing the first release candidate. Then the [Linus Torvalds](#) pulls fixes sent by the same maintainers. Roughly every week a new release candidate is released. After some release candidates (usually seven) a new major release is done.

During the stabilization phase the kernel maintainers prepare patches for the next merge window, so that they get some testing before being sent to [Linus Torvalds](#). For this testing there is another kernel tree called *linux-next* currently maintained by Stephen Rothwell. It's basically a preview of the next kernel release used for automatic and manual testing.

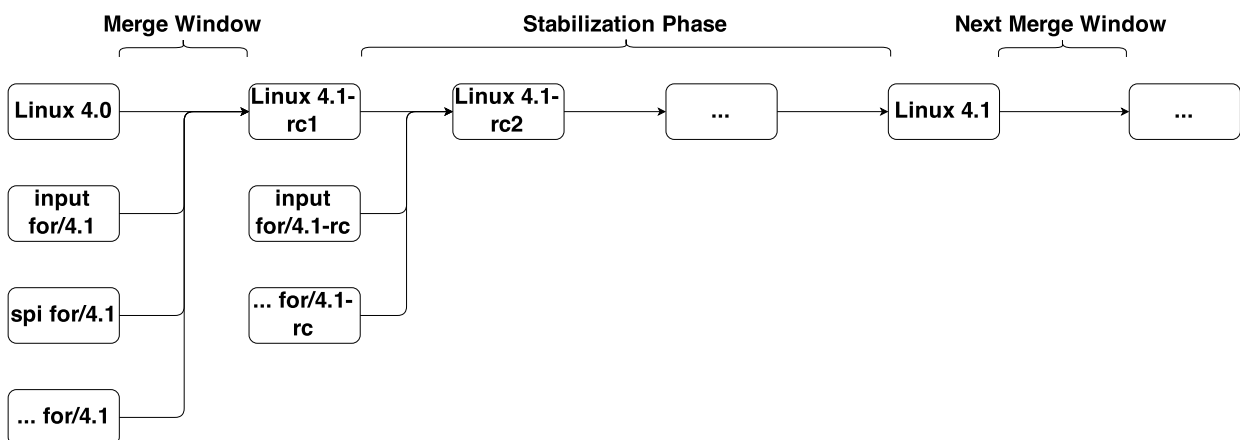


Figure 4.2.: Kernel-development flow



version	release date	version	release date
...	...	3.12	03.11.2013
4.1	22.06.2015	3.11	02.09.2013
4.0	12.04.2015	3.10	30.06.2013
3.19	08.02.2015	3.9	28.04.2013
3.18	07.12.2014	3.8	18.02.2013
3.17	05.10.2014	3.7	10.12.2012
3.16	03.08.2014	3.6	30.09.2012
3.15	08.06.2014	3.5	21.07.2012
3.14	30.03.2014	3.4	20.05.2012
3.13	19.01.2014	...	...

Table 4.1.: Major Linux Kernel Releases

The next section will describe which N900 hardware components are not yet supported in the mainline kernel.

## 4.1. N900 kernel support

Before this thesis started the N900 already had quite good kernel support. The first kernel, which was fairly usable with a [DT](#) based boot was 3.16, which has been used for [Debian](#) 8.0 „Jessie” (frozen before this thesis was started). [Table 4.2](#) contains an entry for most chips built into the Nokia N900. The *Driver* column specifies the initial kernel support for the chip’s functionality, while *DT support* specifies when the driver received support for being initialized from [DT](#). The *N900 DT file* column specifies when the chip or it is sub-module has been added to the N900’s [DT](#) file.

As visible in [Table 4.2](#) the kernel supports all of the critical hardware components ([CPU](#), memory, serial) making the effort to work on other drivers endurable, assumed, that one has access to a serial port for receiving early debug messages.

**Display & Touchscreen:** Apart from critical hardware components, the [OMAP](#) processor’s [DSS](#) and the Sony ACX565AKM panel are supported. Thus the display can also be used for debugging, but it has two disadvantages. First of all only problems occurring after the display has been initialized can be debugged. Since the controller needs many other resources ([GPIOs](#), [I2C](#), regulators, ...) it is initialized quite late during the boot process, though. Assuming, that the error occurs after the display has been initialized, many error messages are too long to fit on the screen and thus are cut off. Since the kernel stops after fatal error messages and the kernel handles all device inputs and outputs, it is not possible to copy any fatal error messages either. Additionally there is support for the TSC2005 touchscreen controller and the keypad matrix making it possible to start a [framebuffer](#) based [X-Server](#) providing a basic graphical environment. However, it does not support any 2D or 3D acceleration, since the [DSS IP-Core](#) only provides the capability to fetch data from memory and send it to a display connected via a serial or parallel bus. 2D and 3D acceleration requires support for the PowerVR SGX [Graphical Processor Unit \(GPU\)](#). Unfortunately, the [GPU](#)’s specification is not publicly available.

**MMC:** Also supported are the [OMAP](#)’s [MultiMediaCard \(MMC\)](#) slots, which are used for an internal 32GB eMMC chip and a  $\mu$ SD card slot on the N900. Thus one can simply put the [Linux distribution](#)

#### 4. Linux Kernel

Chip	Functionality	Driver	DT support	N900 DT file	Comments		
TI OMAP3430	Cortex A8 CPU	2.6.22	N/A	3.13	modern Linux kernel interface for graphics used for eMMC and $\mu$ SD		
	IOMMU	2.6.31	3.15	3.15			
	DSS DSS	2.6.33	3.15	3.15			
	↳DRM support	3.3	N/A	N/A			
	MMC	2.6.30	3.4	3.13			
	WDT	2.6.19	3.8	3.13			
	SSI	3.16	3.16	3.16			
	I2C	2.6.19	3.3	3.13			
	SPI	2.6.23	3.4	3.13			
	USB	2.6.28	3.7	3.13			
	UART	2.6.37	3.3	3.13			
	GPIO	2.6.38	3.4	3.13			
	RNG	3.13	3.16	N/A			
	ISP ISP	2.6.39	-	-			
	TMS320 C64x DSP	-	-	-			
	PowerVR SGX530 GPU	-	-	-			
	Temp. Sensor	-	-	-			
	TI TWL5030	Core	2.6.28	3.3		3.13	OMAP companion chip
		Audio	2.6.33	3.7		3.13	
		Vibrator	2.6.33	3.7		3.13	
Voltage Regulator		2.6.30	3.3	3.13			
RTC		2.6.28	3.3	3.13			
WDT		2.6.31	3.9	3.13			
Keypad		2.6.32	3.14	3.15			
Power Button		2.6.30	3.14	3.14			
Analog Digital Converter		2.6.39	3.15	3.16			
Camera Button		N/A	N/A	3.13	connected via GPIO		
Taos TSL2563		2.6.33	3.14	3.15	light sensor		
TI LP5523		2.6.37	3.13	3.13	LED controller (RGB, keyboard)		
Sony ACX565AKM	2.6.35	3.15	3.15	display controller			
TI TSC2005	2.6.39	3.16	3.16	touchscreen controller			
TI WL1251	2.6.31	3.15	3.16	WLAN controller, requires FW			
Si4713	2.6.32	3.19	3.19	FM transmitter			
TPA6130a2	2.6.33	3.13	3.13	headphone amplifier			
TLV320AIC3X	2.6.25	3.13	3.13	audio codec			
Virtual Sound Card	2.6.34	3.16	3.16				
Proximity sensor	N/A	N/A	3.13	connected via GPIO			
Modem	3.16	3.16	3.16				
	McSAAB protocol	3.16	N/A	N/A			
	↳PhoNet protocol	2.6.28	N/A	N/A			
	CMT speech protocol	-	N/A	N/A			
TI NL5350		N/A	N/A	N/A	accessible via Modem		
Battery		3.8	3.17	3.19			
bq27200		2.6.28	2.6.28	3.13	battery fuel gauge		
isp1707a		2.6.37	3.14	3.15	USB identification		
bq24150a		3.8	3.14	3.15	battery charger		
OneNAND		2.6.28	3.8	3.13			
LIS302DL		2.6.32	-	-	accelerometer		
ADP 1653		3.1	-	-	LED controller (flash light)		
STM VS6555		3.4	4.0	-	front camera sensor		
ET8EK8		-	-	-	main camera sensor		
AD5820		-	-	-	auto-focus		
BCM2048	Bluetooth	-	-	-			
	FM radio	3.14	-	-			
IR transmitter		3.7	-	-			

Table 4.2.: N900 Linux Kernel Support (Status: Linux Kernel 4.0)

on one of them instead of being dependent on the bootloader putting everything into the [Random Access Memory \(RAM\)](#) or using network based boot over the [USB](#) interface / [WLAN](#).

**Audio:** Also working is the sound subsystem, which consists of a headphone amplifier (TPA6130a2), two digital-audio-data-serial interfaces (tlv320aic3x), [OMAP's Digital Audio Interface \(DAI\)](#) known as [Multichannel Buffered Serial Port \(McBSP\)](#), as well as some N900 specific glue code.

**Power Supply:** The 3.16 kernel also has basic support for accessing power supply related properties, such as battery capacity or current voltage. In addition there is a driver for the charging chip, which is automatically enabled depending on the [USB](#) state.

**Modem:** Partially working is the modem, which is connected via a serial bus from [Texas Instruments](#), which is called [Synchronous Serial Interface \(SSI\)](#). In the kernel it is supported in the subsystem of the bus' successor named [Highspeed Serial Interface \(HSI\)](#) (which is a [MIPI](#) standard). The code in Linux 3.16 is enough to control the modem, do data connections, exchange short messages and initialize voice calls. Once initiated the voice call is silent, though, since the audio data is not transmitted over the already supported [PhoNet](#) protocol. Instead it uses its own protocol on a second channel.

**Bluetooth:** Also unsupported is the Bluetooth module BCM2048. The old driver written by Nokia is inadequate for the mainline kernel, since it does not support [DT](#) and violates layer policies: The driver is a combination of a BCM2048 Bluetooth and an [OMAP](#) serial port driver. Instead the BCM2048 driver should use the existing [OMAP](#) serial driver via the kernel's serial [API](#).

**Cameras:** Both cameras, the main cameras auto-focus and the flash-light controller are currently unsupported. The 3.16 kernel contains a driver for the [OMAP](#) processor's part called [ISP](#), which takes care of post-processing the image sensors data. It also contains a driver for the front camera sensor. Both drivers are currently lacking support for [DT](#) based initialization, though. Drivers for the main camera and related chips (e.g. auto-focus) are missing. In addition the current kernel architecture was not designed for having two sensors connected to the same [ISP](#) input port as implemented on the N900.

**Miscellaneous:** Apart from the mentioned components there are already a few other supported chips, which are not important for basic usage: FM transmitter (Si4713), RGB led and keyboard backlight driver (LP5523), ambient light sensor (TSL2563) as well as [WLAN](#) (WL1251).

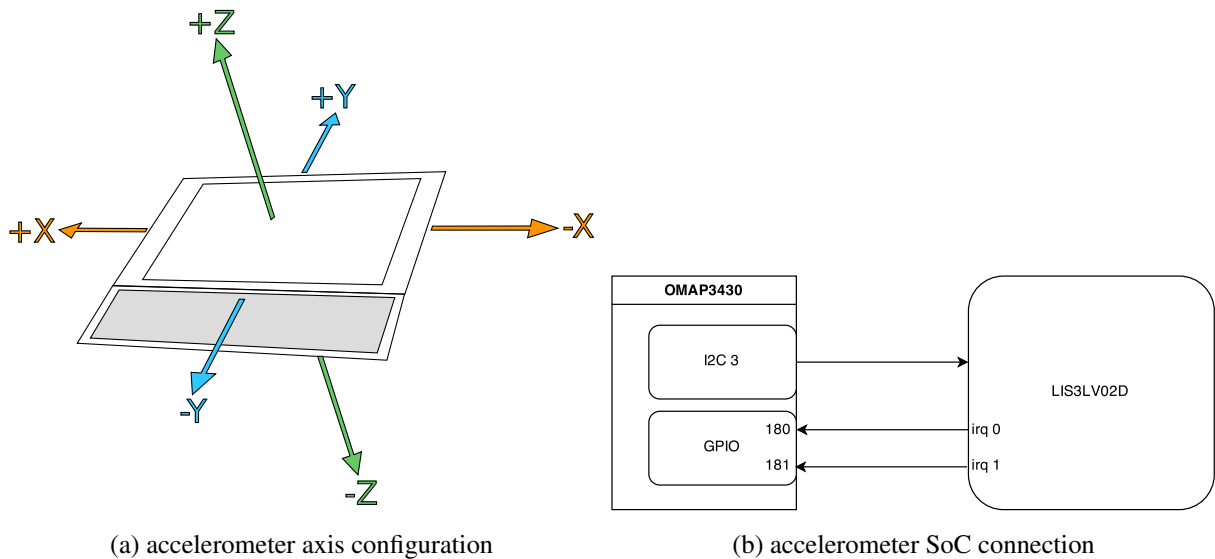
Last but not least it should be mentioned, that the [OMAP](#) processor built into the N900 suffers from a hardware bug, which crashes every binary. There is a workaround in the kernel, which is currently disabled in Debian, since it negatively affects other platforms. This is something that must be fixed for proper out of the box experience.

So summarized most of the really important hardware is already supported since Linux kernel 3.16, main problems are missing speech support and the hardware bug. Also missing is support for Bluetooth and Cameras, which would be nice to have, as they are quite common on modern smartphones.

## 4.2. Accelerometer

One of the simple additions is the accelerometer of the N900, which is already supported by the deprecated board-code based boot, but not yet using [DT](#) based booting. The accelerometer built into the N900 is a

#### 4. Linux Kernel



LIS3LV02D, which is connected to the phone's SoC via one of its I2C ports. As illustrated in Figure 4.3b there are additional interrupt lines provided by the accelerometer, which are connected to GPIO pins of the SoC. These are triggered by the accelerometer when a specified threshold value is reached for one of its channels.

The accelerometer of smartphones are mainly used to detect the device's orientation (e.g. tilt) [34]. This detection is possible, because the accelerometer measures the acceleration of gravity. Apart from that it can be used to give a rough estimate of movement by (e.g. when a GPS signal is lost in a tunnel) [33].

The lis3lv02d has already a Linux kernel driver and is already supported by the soon to be removed N900 board code. So supporting the accelerometer in Debian basically can be established by adding DT support to the lis3lv02d driver. From that task 90% have already been done in the mainline kernel at the beginning of this thesis.

The DT conversion did not expose some of the lis3lv02d's features used on the N900, though. Specifically the second wakeup unit could not be described and the axis could not be inverted. This is a problem, since the accelerometer placement in the N900 does not default to the desired axis configuration as depicted in Figure 4.3a. In addition to the driver changes another patch is needed to update the lis3lv02d's DT binding specification, so that the driver implements the correct binding. Last but not least the N900's DT source file had to be updated, so that it advertises the accelerometer to the kernel.

All patches have been added to the Linux 4.1 kernel and can be seen in Appendix C.

After this introduction into what is needed for mainlining support for a simple sensor, the thesis will focus on more complex things starting with one of the most important components for a smartphone - the modem used for Universal Mobile Telecommunications System (UMTS) and Global System for Mobile Communications (GSM) connections.

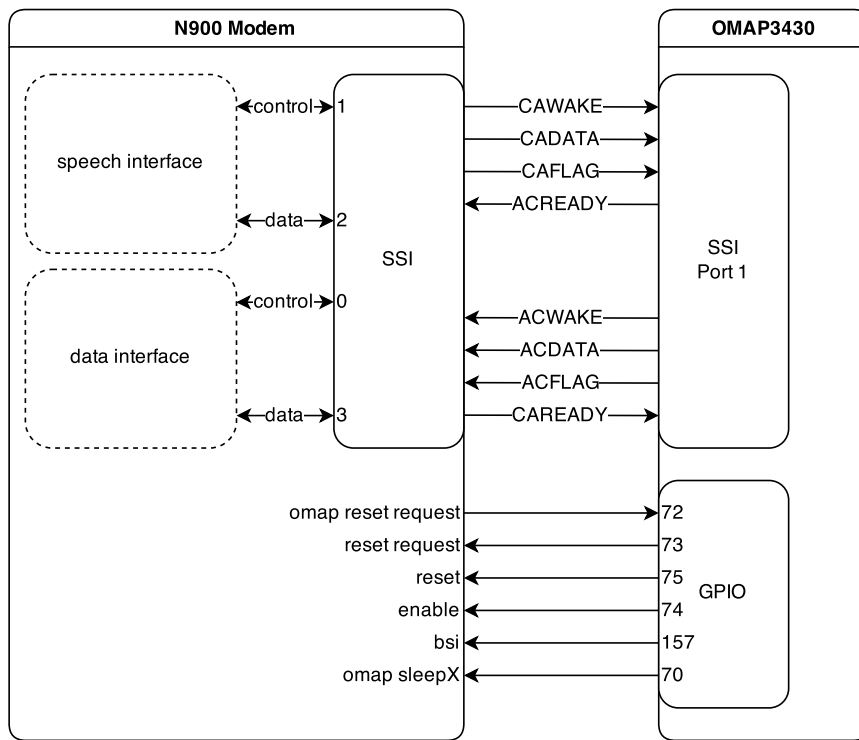


Figure 4.4.: Modem interconnection

### 4.3. Modem

In opposite to some of the [OMAP](#)'s competing [SoCs](#) it does not come with a built-in modem. Instead it provides a high-speed serial interface called [SSI](#). This port has been used by Nokia to connected their baseband processor as shown in [Figure 4.4](#). Note, that this is a very simplistic view, since the modem actually consists of multiple chips on the N900's [PCB](#) (e.g. it has a module providing 128 MB [RAM](#) and 128 MB flash memory making it more or less independent from the remaining system).

Even more simplified the modem could be described as feature phone connected via a serial port having its power button(s) connected via [GPIOs](#). For supporting the modem multiple things had to be implemented in the mainline kernel.

When Carlos Chinaea started to push support for the [SSI](#) interface to the mainline kernel, a standardized successor of the protocol already existed, namely [HSI](#). Thus a new framework was introduced by Carlos, which provides some generic functionality. He also provided initial patches providing support for the [OMAP3](#)'s [SSI IP-Core](#) [7].

After a couple of reworks later [Linus Torvalds](#) pulled the [HSI](#) framework into the 3.4 kernel [37] - without the [OMAP3-SSI](#) support, which was still being worked on by Carlos. Afterwards Nokia completely shut off their Maemo/MeeGo work and Carlos had no more time to work on the [SSI](#) support [39].

Following that I took over Carlos' subsystem at the end of 2013 to continue his work by adding mainline support for the [OMAP3-SSI IP-Core](#). At that point the internal kernel structures have improved quite a bit and [DT](#) had been introduced, so that it took almost one more year to get the drivers in shape.

#### 4. Linux Kernel

Before looking at the protocols used to exchange data with the modem it is important to understand some of SSI's (and HSI's) design concepts. As visible in Figure 4.4 there are eight signals connecting the SSI port of the modem with the SSI port of the OMAP. The signals can be divided into two groups, one is used to send data from the modem (also known as *cellular die*) to the main processor (also known as *application die*) and one group is used to send data in the opposite direction.

The four signals contained in each group are named DATA and FLAG, which are a combined data and clock signal. Additionally there is a READY flag, which is used by the receiving side to indicate, that it is ready to receive. Last but not least there is a WAKE signal, which can be used to wakeup the receiving side before sending data. This signal is optional in the HSI standard, but available on the OMAP and used by the N900.

While HSI is specified for data rates up to 200 Mbps [16], the SSI controller, which is part of the OMAP3 only reaches data rates of 55 Mbps according to comments in Nokia's original driver.

In addition to the physical interface the (non-public) HSI standard also describes the data link layer, which supports 1-8 logical channels for data transmit and receive respectively. It also defines two different modes to send data (*frame-based* vs *stream-based*) and three different flow modes (*synchronized*, *pipelined* and *real-time*). The logical channels and the data exchange modes are implemented in OMAP's SSI IP-Core and data exchange happens using DMA and interrupts.

As visible in Figure 4.4 the SSI connection provides four logical channels for the modem communication. Two of those channels are used for generic communication with the modem. The protocol used with those channels is known as McSAAB and uses one of the channels as control channel and the other as data channel. The other two channels are used for exchanging speech data with the protocol also using one channel as control channel and one for data exchange.

The mainline kernel got support for the McSAAB protocol together with the low level OMAP SSI driver in 3.16, since it was used to test the SSI implementation. The kernel's McSAAB driver provides access to the modem by providing a network device prefixed with *phonet*. Instead of using the standard Ethernet media type it sets a custom PhoNet media type, which is handled by its own protocol handler named *af\_phonet* in the kernel. Figure 4.5 depicts the previously described modem driver architecture.

The PhoNet protocol (supported in the mainline kernel since 2.6.28) is a simple packet based binary protocol, which is an abstraction for the modem interconnect. The protocol has support for channel multiplexing and asynchronous notifications and can be forwarded over USB using a gadget driver on the phone (also part of the mainline kernel since 2.6.28) and a matching USB host driver (mainline since 2.6.31). The PhoNet driver also takes care of creating a second network device for General Packet Radio Service (GPRS) connections. In opposit to the PhoNet network device the GPRS network device is used with Internet Protocol (IP) based packets. Since the device is only created after the authentication has already been done and GPRS systems usually use Dynamic Host Configuration Protocol (DHCP) to provide IP addresses, using the GPRS network device behaves very similar to e.g. a WLAN network device.

Currently not handled by the kernel is the modem initialization, which requires enabling and disabling of the GPIOs seen in Figure 4.4. The GPIOs are currently simply exported to the userland and must be taken care of by the userland software managing the modem.

On top of PhoNet another protocol is involved in the modem communication, which is Nokia's ISI protocol. This protocol is also not handled in the kernel, but forwarded to the userland encapsulated in PhoNet frames and is supposed to be handled there.

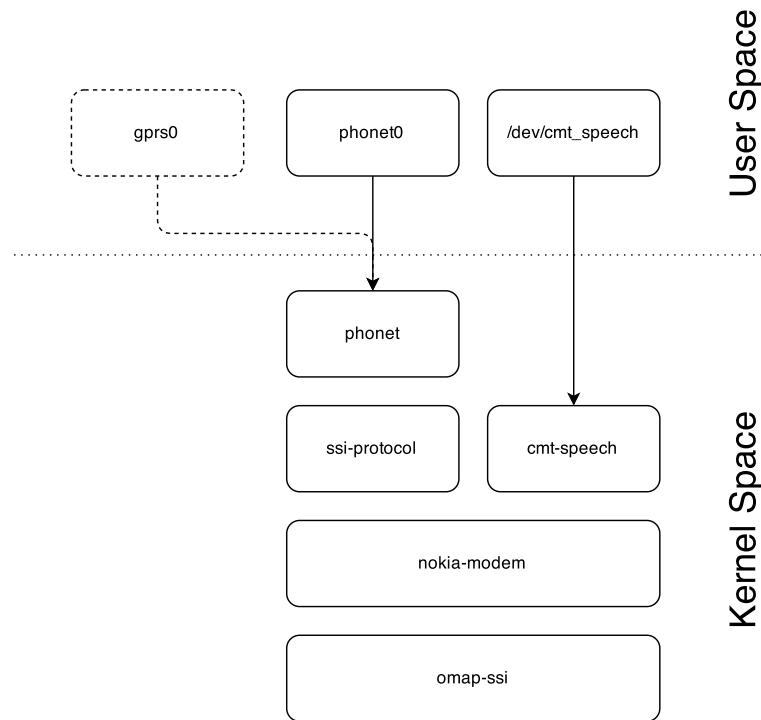


Figure 4.5.: Modem protocol stack

So the current kernel supports a wide range of the modem's functionality when used with the right userland software. Not yet supported though is exchanging voice data with the modem, as that part does not happen using the [PhoNet](#) protocol or the [McSAAB](#) protocol for that matter. Instead a custom SSI protocol is used, which encapsulates small frames of voice data.

Like [McSAAB](#) the speech data is exchanged in frames. Nokia's original driver exports a simple character device `/dev/cmt_speech` for userland access to the speech data. This device simply outputs received frames by reading from the file and sends frames by writing to the file. Some extra functionality is provided by using custom `ioctl`s, e.g. for setting the [SSI](#) wake-line. While Linux usually handles audio data using [Advanced Linux Sound Architecture \(ALSA\)](#), Nokia's developers have decided against using it. The problem is, that [ALSA](#) has been designed with [Pulse-code modulation \(PCM\)](#) interfaces in mind. The speech protocol on the other hand is a lossy protocol with data send in packets (with metadata, like e.g. timing information) instead of a stream. Apart from that the speech protocol used over SSI has strict timing requirements (< 20ms) [38]. This timing constraint is most likely directly passed on by the modem, which must send e.g. 160 samples every 20ms if the full rate codec is selected [32, p. 46].

The difference to a standard PCM connection can be seen, when the modem loses its coverage during a voice call. The PCM voice link provided by most modems would still provide data samples (usually simply containing silence). The speech interface provided by the N900 on the other hand does no longer send any voice packets [38], since it does not receive anything.

During this thesis Nokia's original driver has been updated to use the new Linux kernel [APIs](#) and integrated into the [DT](#) initialized `nokia-modem` driver. Since the complete driver is over 1000 lines of code, it has not been added to the Appendix, but can be found on the DVD enclosed. The changes have been pulled by Linus Torvalds into the 4.1 Linux kernel, so that voice calls could be established with it.

In the next section the ARM CPU bug mentioned in the chapter's introduction will be analysed further with the aim to find a workaround acceptable for Debian.

### 4.4. ARM Errata 430973

Apart from missing hardware support, there may also be problems with hardware bugs. These are especially problematic, if they are bugs in the CPU, since that will affect normal code processing. The N900 has a couple of them in its ARM CPU core.

One of them, named ARM Errata 430973, is especially annoying, since it is triggered quite often by normal code execution, if the code is using the CPU's Thumb mode. This mode is a subset of the ARM instruction set, which further reduces (ARM is based on a Reduced Instruction Set Computing (RISC) design) the normal instruction set. The advantage of a smaller instruction set is a small memory footprint resulting in better cache usage. Modern compilers usually generate code switching the instruction set at runtime to gain the better cache usage without losing the features from the full instruction set. While the original OS shipped with the N900 does not trigger the bug, since the software is not compiled using the Thumb mode, this is not the case for other Linux distributions, which normally enable it. For example a armhf Debian userland usually has a segmentation fault after being used for a couple of minutes.

For triggering the bug some code fragment must contain an ARM/Thumb interworking branch, which would normally switch between ARM and Thumb mode at runtime. As previously mentioned this is used on modern systems to exploit the features of both modes. If a code fragment containing such an interworking branch is replaced with other code at the same virtual address the Cortex A8 does not recover from the stale interworking branch prediction resulting in code being executed in the wrong mode. The code replacement at the same virtual address is easily triggered by the kernel re-mapping virtual to physical addresses when the process is switched.

A workaround for this bug has been implemented in the Linux kernel in 2009 [23] by flushing the branch target cache at every context switch on ARMv7 systems. Unfortunately its usefulness is quite limited on kernels supporting more than one ARM platform, since flushing the cache may be quite expensive and is not required on unaffected systems. So if a kernel with the workaround is started on an unaffected system it will have a useless performance drop. With the number of affected systems decreasing its thus a hard decision to enable the workaround for generic ARM kernels used by most Linux distributions.

After some discussion with the ARM and OMAP subsystem maintainers a patch has been prepared by Russell King, which avoids the branch target buffer flush on non-Cortex A8 based CPUs [17]. As a result the workaround can be enabled without negatively affecting performance on most unaffected systems.

Before the branch target buffer flush opcode becomes operative it must be enabled by setting the IBE bit, though. Doing this is platform specific and must be done as soon as possible, since the kernel may be compiled in Thumb mode itself. Thus the IBE bit should actually be set by the bootloader on affected systems. A patch for that has been written for U-Boot for all supported OMAP3 based boards [24].

While this has also been done for The Nokia N900's U-Boot port, it should also be set in the kernel, since U-Boot is only used as optional chain loaded bootloader and Nokia's original bootloader can't be fixed easily. Thus the bit is also set in a platform quirk for the N900 if the workaround is enabled in the kernel configuration [31].



While the implemented changes are enough to allow [Linux distributions](#) enabling the workaround, there is one more possible optimization. The kernel can unconditionally enable the cache flushing for Cortex A8 based systems and require, that the state of the IBE bit is set correctly by the bootloader (enabled if the system is affected by ARM Errata 430973 and disabled for unaffected systems). Additionally the N900 IBE quirk can be enabled unconditionally, since all released N900s are affected by the bug.

With the most important bits working the next section will have a look at kernel support for the N900's Bluetooth module.

## 4.5. Bluetooth

On the Nokia N900 Bluetooth support is handled by Broadcom's BCM2048, which is mainly accessed via one of the OMAP's [UART](#) ports as visible in [Figure 4.6](#). Apart from Bluetooth, the BCM2048 also supports receiving [Frequency modulation \(FM\)](#) radio. While the Bluetooth functionality is handled via the [UART](#) port, [FM](#) radio functionality is handled via [I2C](#). Apart from that the audio processing of the Bluetooth and [FM](#) receiver submodules is handled differently: The [FM](#) radio data output is sent directly to the audio codec, while the Bluetooth audio data is exchanged bidirectionally with one of the OMAP's [McBSPs](#).

Since the Bluetooth and the [FM](#) radio interface are basically independent from each other, they are handled by different drivers in the Linux kernel. Currently (Linux 4.1) there is a driver for the [FM](#) radio part available in staging named *bcm2048-radio*, which handles the [I2C](#) communication with the BCM2048 and exposes the module's functionality using the [Video for Linux 2 \(V4L2\)](#) radio [API](#). There will be no further analysis of the BCM2048's [FM](#) radio part in this thesis.

The primary part of the BCM2048 is currently not supported in the mainline kernel and, in opposit to the modem drivers, a simple porting of Nokia's original driver is not possible. The main problem of Nokia's driver is a missing abstraction between the OMAP's [UART](#) port and Broadcom's Bluetooth chip. The original driver handles both, the [UART](#) port and the Bluetooth communication. This missing hardware abstraction is problematic, since it prevents reusing the BCM2048 driver if its connected to another [UART](#) module and results in code duplication. Apart from that the BCM2048 module must be described separately from the [UART IP-Core](#) in the [DT](#) and drivers are normally hooked to nodes in the [DT](#). Thus initializing such a driver is hard in the mainline kernel.

After some discussion with developers from Broadcom working on mainline support for Broadcom's Bluetooth products and the Linux kernel's Bluetooth maintainer it became clear, that the BCM2048 built into the N900 has been customized by Nokia. As a result Nokia should be mentioned in the compatible string, since other vendors may have their own customizations and are probably not using Nokia's extensions.

The customization also has some implications regarding the firmware file. While Broadcom would probably release a distributable firmware file for the BCM2048 as they did for other Bluetooth chips, they can't do it for the customized variant. Thus such a firmware file would have to be released by Nokia's smartphone division, which has been taken over by Microsoft in the meantime.

Further analysis of the original driver reveals, that it is basically using the standard Bluetooth [UART Host Controller Interface \(HCI\)](#) protocol as specified in [12] also known as [H:4](#) with some minor additions, which probably gave the protocol its name *H4+*.

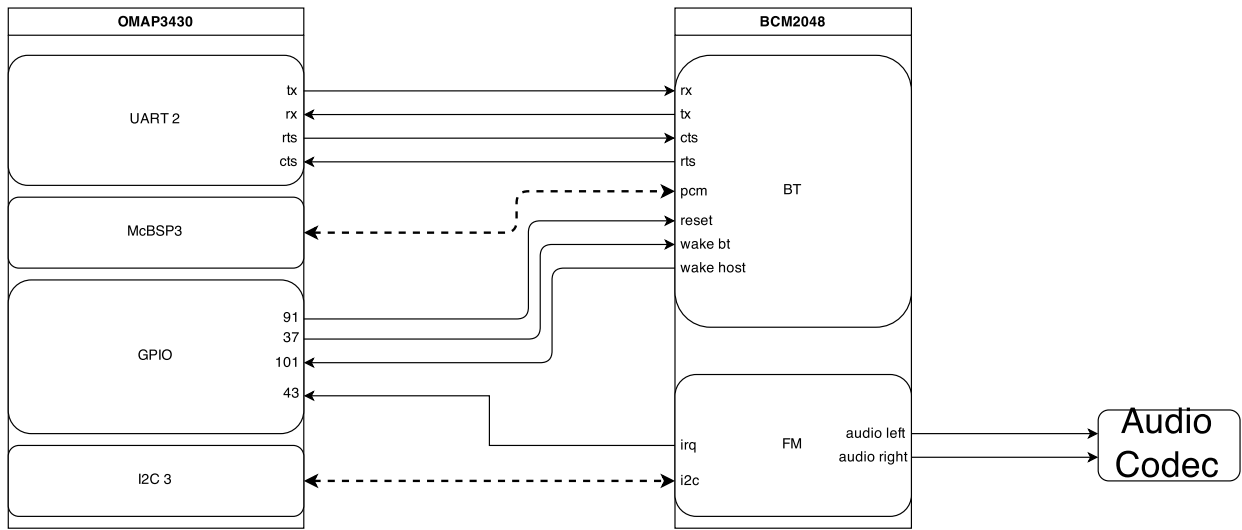


Figure 4.6.: BCM2048 setup

Packet Type	Packet Indicator	H4 Standard
HCI Command Packet	0x01	yes
HCI <a href="#">Asynchronous Connection-Less (ACL) Data Packet</a>	0x02	yes
HCI <a href="#">Synchronous connection-oriented (SCO) Data Packet</a>	0x03	yes
HCI Event Packet	0x04	yes
HCI Negotiation Packet	0x06	no
HCI Alive Packet	0x07	no
HCI Radio Packet	0x08	no

Table 4.3.: Nokia H4+ Packet Types

While the [H:4](#) protocol provides four different packet types Nokia's BCM2048 driver uses three more packet types, which are not part of the standard, as visible in [Table 4.3](#). Apart from that the driver must take care of the additional [GPIOs](#) for waking up the Bluetooth chip before sending data to it.

For developing an open driver it was first necessary to understand what the additional packet types are used for. Checking Nokia's driver the Negotiation packet is the first exchanged packet and is responsible for setting up the BCM2048's baud rate, protocol and provides system clock speed information. Before receiving this packet the BCM2048 can be accessed using a baud rate of 120000.

The alive packet is a simple alive check, which is answered by the BCM2048 to verify, that the UART connection is working. Testing revealed, that its the only packet, that can be sent before Negotiation without triggering an hardware error event.

Last but not least the radio packet is probably used for setup of the [FM](#) radio part built into the BCM2048. Nokia's original driver does not handle it except for sending a single message as part of the firmware.

Since all of the extra packets are only sent during the device initialization and part of the firmware file, only the receiving side needs special handling. Marcel Holtmann, who is currently maintaining the Bluetooth subsystem in the Linux kernel, changed the subsystem making it possible for Bluetooth UART drivers to register their own packet type handlers and prepared a simple driver, which implements the above information about Nokia's H4+ protocol [14, 13].

With the restructured [H:4](#) code its possible to reuse its functions for splitting the incoming data stream from the UART into individual packets as required for further processing in the Linux kernel's network subsystem [30, p. 298-310]. All standard H4 packets are simply passed to the Bluetooth subsystem in the kernel, which does any further required processing.

Not yet implemented in Marcel Holtmann's template driver is the UART handling and the [GPIO](#) handling. As shown in [Figure 4.6](#) there are three important [GPIOs](#): *reset*, *wake-bt* and *wake-host*. The *wake-host* [GPIO](#) is needed to allow the OMAP's [UART](#) transitioning into low power states. While it can be waken by simply receiving data, it would loose the first few bytes this way, since it needs some time to wake up. Instead the *wake-host* [GPIO](#) is set shortly before data is sent by the BCM2048.

The *wake-bt* [GPIO](#) basically does the same thing in the opposite direction. It must be set by the kernel before sending data to the BCM2048 and should be unset once all data has been sent.

Last but not least the device can be reseted using the *reset* [GPIO](#). The reset procedure must be performed before initial use and after hardware error events are received.

Last but not least its important to correctly setup the RS232 settings. The Bluetooth specification [12] specifies, that devices should use the common [8-N-1 format](#) and make use of [Ready to Send \(RTS\)](#) and [Clear to Send \(CTS\)](#) signals for flow control. Additional information like baud rates can be taken from Nokia's original driver: After reset the chip initially uses a baud rate of 120000 ( $\approx 15$  kB/s). Then its increased to 921600 ( $\approx 922$  kB/s) after the negotiation took place. Last but not least its further increased to 3692300 ( $\approx 1.8$  MB/s) once the firmware has been transmitted.

For implementing a mainline kernel driver another problem is the [DT](#) binding. Currently the kernel has not a single [UART](#) attached driver with a [DT](#) binding, which could be used for reference. When this topic has been brought up at the [Linux kernel mailing list \(LKML\)](#), two different binding styles have been proposed.

## 4. Linux Kernel

The first proposal adds a sub-node as child of the [UART](#) node, while the second proposal puts the [UART](#) attached device somewhere else (usually at the root level of the tree) and adds a link to the [UART](#) device. Both proposals support all use cases currently expected by the kernel developers on the [LKML](#) and would support loading the BCM2048 driver.

The main argument for using references instead of a parent-child link brought so far can be summarized, that parent-child relationships in [DT](#) so far either means, that the child is connected to the parent using an addressable bus or the parent being a [multi function device \(MFD\)](#) with the children describing individual functions of it. The second argument for using references is, that they are more powerful than a parent-child description, since they allow multiple connections, while there can be only one parent.

On the other hand the arguments for binding [UART](#) attached devices using a parent-child style assume, that the parent-child dependency show the chip's "main" interface. The observation, that it is used for addressable buses and [MFD](#) devices is the result of the kernel not supporting other "main" interfaces so far. Also while a reference styled binding is more powerful in its expressiveness it makes parsing the "tree" harder. In theory everything could be using just references instead of parent-child connections. Currently the kernel uses parent-child connections for automatic runtime power management: If a child is supposed to be accessed, the kernel must enable its parent first. A discussion around this problem can be found in the thread following the mail at [5].

A simple driver, which extends the skeleton one from Marcel Holtmann has been prepared for the mainline kernel as part of this thesis and can be found on the attached DVD, but so far the BCM2048 only sends hardware error events making the driver useless in its current state. It will be further worked upon and is expected to be ready in time for the Linux 4.3 kernel.

In the next section the cameras and the OMAP's image signal processor will be analysed.

### 4.6. Camera Subsystem

Another component of the N900 not yet supported in the mainline kernel is the camera subsystem. The N900 has two different cameras, which are built to the phone's front (vs6555) and back (ET8EK8). As shown in [Figure 4.7](#) both cameras are connected to the same port of the OMAP using a multiplexer ([GPIO](#) controlled switch). In addition to the interface for the pixel data both sensors can be configured via I2C.

Both sensors are supplied with a reference clock from the OMAP, which is fed into a built-in [Phase Locked Loop \(PLL\)](#) to generate the clock used for sending pixel data. In addition both sensors have an enable pin, which can be used to send them into low power states. Unfortunately the [GPIO](#) used for enabling the front camera is also used to setup the video bus multiplexer.

Currently the mainline Linux kernel has drivers for the OMAP's [ISP](#) module and a generic driver for [Standard Mobile Imaging Architecture \(SMIA\)](#) compatible camera sensors, which can be used for the vs6555. Missing are drivers for the ET8EK8, which is not fully [SMIA](#) compatible and for the video bus multiplexer. Once the basic support for the ET8EK8 is working, an additional driver would be needed for the AD5820, which drives the motor controlling the lenses.

Both sensors are configured to use the [Camera Serial Interface \(CSI 1\)](#) protocol, which is also known under the name [Compact Camera Port 2 \(CCP2\)](#), though with slightly different clock configuration.

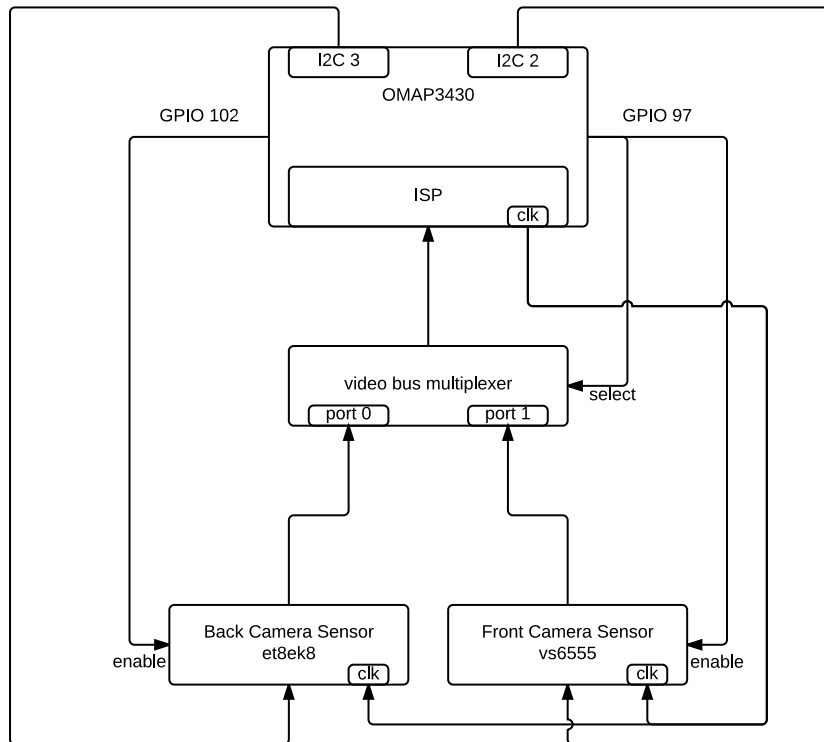


Figure 4.7.: Camera setup

A solution to describe video pipelines in DT has been discussed in the kernel community. The result was the creation of a generic graph binding description, which can be found in *Documentation/device-tree/bindings/graph.txt*. In opposit to simple [phandles](#), which allow simple directed graphs, it allows to add bi-directional links between two nodes in the DT. It's supposed to be used for connections, which cannot be inferred from DT's parent-child relationships.

While the graph binding makes it possible to describe, that there is a connection, it does not provide any information about the link between the two devices. For this an extended description has been created in *Documentation/devicetree/bindings/media/video-interfaces.txt*. It extends the common graph binding by video interface specific properties.

To demonstrate how the generic video interface binding can be applied a simplified DT description for the N900's display connection can be seen in [Listing 4.1](#). Basically a port node is added to the device, which may have an optional reg value for identifying the port number. Since this may require to setup correct size information using `#address-cells` and `#size-cells`, the port nodes may be grouped together in a ports node. Inside of the port node an endpoint node is added, which contains a [phandle](#) to the remote endpoint node.

Listing 4.1: Simplified Nokia N900 Display Connection DT excerpt

```

1 &dss {
2     status = "ok";
3
4     /* ... additional properties ... */
5
6     ports {
7         #address-cells = <1>;
8         #size-cells = <0>;

```

## 4. Linux Kernel

```
9
10     port@1 {
11         reg = <1>;
12
13         sdi_out: endpoint {
14             remote-endpoint = <&lcd_in>;
15             datapairs = <2>;
16         };
17     };
18 };
19 };
20
21 &mcspi1 {
22     acx565akm@2 {
23         compatible = "sony,acx565akm";
24         spi-max-frequency = <6000000>;
25         reg = <2>;
26
27         /* ... additional properties ... */
28
29         port {
30             lcd_in: endpoint {
31                 remote-endpoint = <&sdi_out>;
32             };
33         };
34     };
35 };
```

Based on this Sakari Ailus started to work on a [DT](#) binding for the generic [SMIA](#) driver of the mainline kernel (*smiapp*), which can be seen as the successor of the one used in Nokia's kernel for the vs6555 sensor. The main work here is the conversion to [V4L2](#)'s asynchronous sub-device registration API, which allows sensor drivers to be registered before the [ISP](#) they are connected to. This is important for the [DT](#) initialized drivers, since the sensor will be probed for when the [DT](#) parser hits the matching [DT](#) node. Previously the [ISP](#)'s sub-devices have been registered via the [V4L2](#) framework after the [ISP](#) has been initialized. In addition support for some device specific properties had to be added. While [DT](#) support for the driver has been added in the 4.0 kernel, neither the driver, nor the [DT](#) binding support the [CCP2](#) protocol used on the N900.

Also worked on by Sakari Ailus was [DT](#) support for the *omap3isp* driver, which handles the OMAP's [ISP](#) based upon Laurent Pinchart's work regarding [DT](#) support for the [ISP](#)'s [Memory Management Unit \(MMU\)](#). With the [DT](#) based initialization working the N900's kernel exposes the [V4L2](#) sub-devices visible in [Figure 4.8](#) to the userspace. In this figure the green boxes (rounded corners) are sub-devices for some specific functionality of the [ISP](#) and the yellow (sharp corners) are userspace accessible video sources/sinks. For accessing the N900 cameras it is important, that at least the *OMAP3 ISP CCP2* sub-device works, since the cameras are both connected to it. It's also important, that the *OMAP3 ISP CCDC* sub-device is working, since it is not possible to access video data directly from the *OMAP3 ISP CCP2* sub-device. At later steps it also important to get the [Auto Exposure White Balance \(AEWB\)](#), [Auto Focus \(AF\)](#) and histogram sub-devices working to provide good images. The only sub-device completely irrelevant on the N900 is the CSI2a block, since nothing is connected to it.

Equally to the *smiapp* [DT](#) binding, Sakari's *omap3isp* [DT](#) binding does not include support for the [CCP2](#) protocol. Thus the amount of work needed to get the N900's cameras working can be summarized as

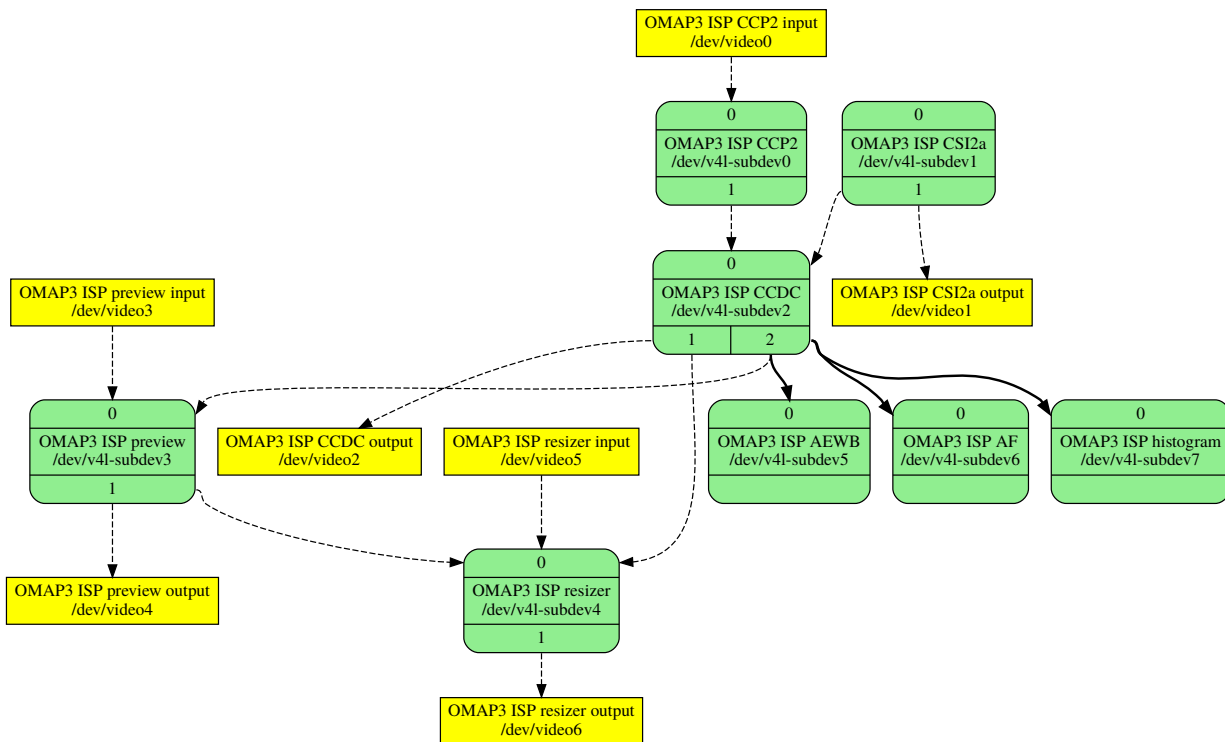


Figure 4.8.: OMAP's image signal processor in Linux

follows:

1. Add [CCP2 DT](#) support to the *omap3isp* driver.
2. Add [CCP2 DT](#) support to the *smiapp* driver.
3. Create new driver for the video bus multiplexer

This will be enough to get the front-camera working. The main camera additionally needs a completely new sensor driver and a driver for the [AF](#)'s motor coil.

In the context of this thesis a driver for the video bus multiplexer and [CCP2](#) support for the *omap3isp* and the *smiapp* driver have been implemented. The result can be seen in [Figure 4.9](#), which depicts the camera setup as seen by the Linux kernel. The shapes have the same meaning as the ones in [Figure 4.8](#). In addition dashed arrows stand for unconfigured connections, while solid arrows stand for configured ones with solid bold arrows being immutable connections.

Unfortunately the video pipeline still contains a bug resulting in no data being received at all. There are many possible reasons for this and the next planned step is checking at hardware level if any data is sent by the sensor using an oscilloscope or logic analyzer. For time reasons this will be postponed after the thesis, though. The existing code can be found on the attached DVD.

The next chapter will have a look at the userland software required to make use of the features provided by the Linux kernel.







## 5. Userland

Apparently a working kernel is not a full Linux distribution. This chapter will describe how the Debian GNU/Linux userland can be prepared for the Nokia N900.

The smartphone has three different persistent memories: A 256MB NAND flash, a 32 GB eMMC flash and a  $\mu$ SD card slot supporting the SD High Capacity standard (SDHC, up to 32GB). The phone is shipped with Maemo installed to the NAND with additional files in the eMMC flash. While the 32GB eMMC flash could be repartitioned to gain some space for another operating system this thesis will focus on installing the userland on a  $\mu$ SD card. Apart from most SDHC  $\mu$ SD cards being faster than the eMMC flash, it is also simpler to debug problems, since the  $\mu$ SD card can be removed from the phone and used with a card reader.

Currently the Debian installer does not support the N900. Instead the image must be created manually. [Appendix A](#) provides a detailed explanation how this is achieved from a running Unix system. This chapter will focus on services required to support specific hardware components of the phone once the base image can be booted on the phone.

After booting the image described in the Appendix, the user will be greeted with a simple console login prompt. After providing the login credentials, one has a fully functional shell available. At the beginning of this thesis, there was a problem with the arrows keys, since Debian's ckbcomp utility, which converts X.org keyboard descriptions into console keyboard descriptions did not fully understand the N900 keyboard description file<sup>1</sup>. This has been fixed by adding support for the keyboards special function key to ckbcomp. Unfortunately the N900's keyboard layout does not provide hardware keys for  (Tab), which is usually used for command completion. Typing each command fully is especially inconvenient on a mobile hardware keyboard. Also missing is a  (Pipe) key, so there is no easy way to redirect the command output into a pager. Thus its not easily possible to use commands with output not fitting to the N900's screen.

### 5.1. Graphical User Interface

Thus the first thing, that should be done is to start the graphical X server and see if everything works as expected. To avoid seeing just a blank screen with a cursor on top, we should first configure an application being started together with the X server, though. For a first try it is advisable to simple start a xterm. Thus the first steps can be seen in [Listing 5.1](#).

Listing 5.1: Starting X-Server together with an XTerm

```
1 echo xterm > .xinitrc
2 startx
```

---

<sup>1</sup><https://bugs.debian.org/789816> - Nokia N900 Keyboard Model is not fully supported by ckbcomp

## 5. Userland

Assuming everything works as expected one should be greeted by a white terminal on top of a black background. Apart from that there should be a cursor visible somewhere on the screen.

It is advisable to calibrate the touchscreen before going on with further customization of the Debian image, since the uncalibrated touchscreen is basically unusable. This can be done by typing `xinput_calibrator -v` into the open terminal window. The calibration software will display a small reticle on the display's top-left corner. Once the reticle has been touched another will be shown in the next corner. After a reticle has been touched in all corners, the application will exit showing the touchscreen configuration looking similar to [Listing 5.2](#). The generated snippet should be copied into `/usr/share/X11/xorg.conf.d/20-ts-calibration.conf`, so that the X server will pick it up during start-up.

Listing 5.2: X.org Touchscreen Configuration

```
1 Section "InputClass"
2     Identifier    "Touchscreen"
3     MatchProduct "TSC2005 touchscreen"
4     Option        "Calibration" "200 3910 3761 180"
5 End Section
```

Once the Touchscreen has been configured, basic keyboard and mouse input should work, so its time to replace the minimal xterm setup with a full desktop environment. Here it is possible to choose from a wide array of window managers, that have been written for Desktop PCs. Since neither the mainline kernel, nor the Debian userland provide any 3D acceleration capabilities for the Nokia N900, [Open Graphics Language \(OpenGL\)](#) using window managers should be avoided, though. For example in the installation process described in [Appendix A](#) the desktop environment from the Enlightenment team has been installed, which has explicit support for mobile devices. The Enlightenment desktop environment can be started using `enlightenment_start`, so the `~/.xinitrc` file should be updated accordingly. After restarting the X server (it will automatically quit by closing the terminal window), Enlightenment will start up with a simple configuration wizard requesting some user specific information (e.g. the user's preferred language) and the option to choose between the desktop and the mobile variant. The mobile variant comes with a desktop and two panels. [Figure 5.1](#) depicts the Enlightenment desktop environment with a split screen setup (the fourth button in the top panel switches between mono, horizontal- and vertical-split) running the web-browser `midori` on the left and Enlightenment's terminal application on the right.

After verifying, that everything works as expected, a display manager can be installed, so that the system boots directly into a graphical user interface. A suitable candidate is `entrance` from the Enlightenment project (not available in Debian) or `lightdm`. It's installation requires a network connection, though, so the next section will deal with that first.

### 5.2. Wireless LAN

The Wireless LAN driver for the `wl1251` built into the Nokia N900 uses the standard framework for WLAN in the Linux kernel. Thus the interface is managed using `wpa_supplicant`, which provides an [inter-process communication \(IPC\)](#) interface for high-level network managers. Possible candidates managing the network interfaces are `network-manager`, `wicd` or `connman`. If the network manager should also take care of the WWAN data connection later, `wicd` should be avoided, since it does not provide such functionality. Similarly there is no support for `ofono` in `network-manager`, which uses `modem-manager` for its WWAN

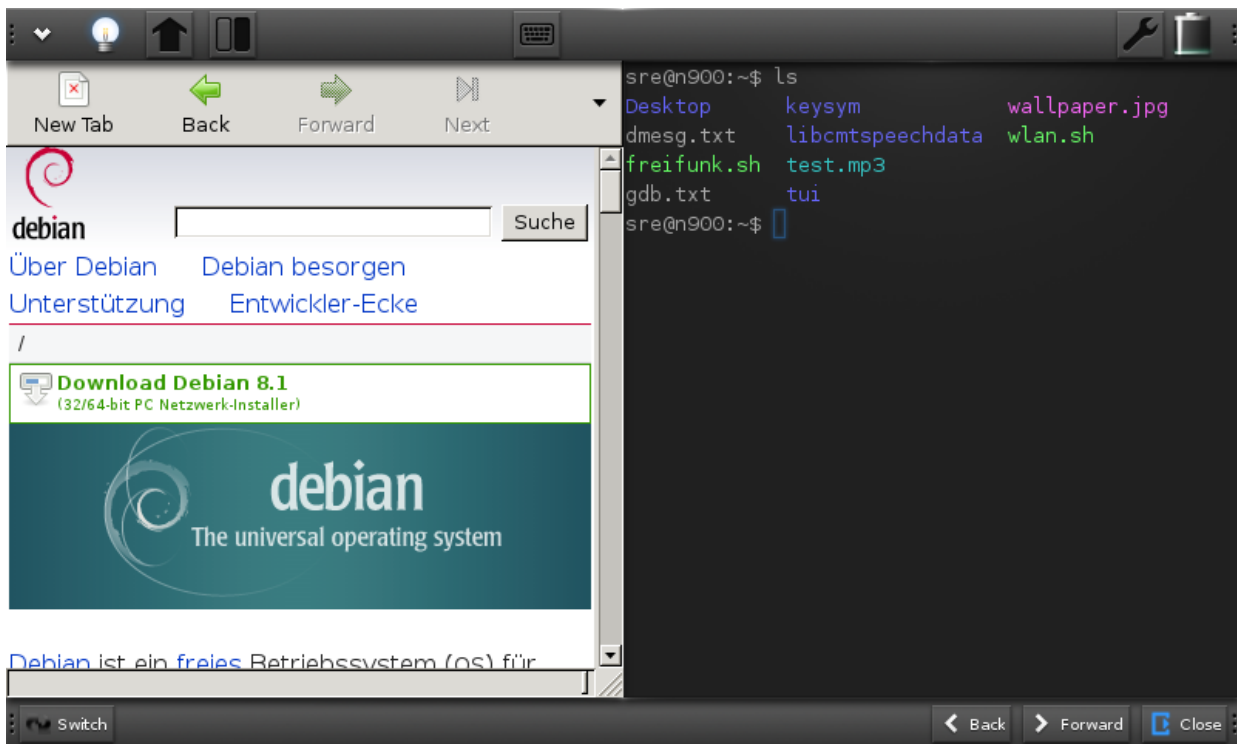


Figure 5.1.: Screenshot from Enlightenment running in vertical split mode

modem capabilities<sup>2</sup>. Thus the best option for handling the data connections is *connman*, which comes with *ofono* and *wpa\_supplicant* support out of the box.

For configuration of the data connections the Enlightenment desktop environment comes with a simple configuration utility. Unfortunately it is currently disabled in Debian's Enlightenment build currently. For this thesis the package has been rebuilt including the *connman* configuration tool. For other desktop environments *connman-ui* should be installed in addition to *connman*. It provides a GTK+ based configuration GUI for *connman*.

### 5.3. The modem

As described in [section 4.3](#), the modem is connected via one of the OMAP's SSI ports. The kernel provides a driver named *ssi-protocol*, which implements the layer 2 protocol for modem communication. It provides a network interface to the userspace, which can be used to send messages to the modem. While this interface would normally be named *ssip* (short for *ssi-protocol*) or *mcsaab* (the protocol's original name was **McSAAB**), it has been named *phonet*. **PhoNet** is the name of the layer 3 protocol used with the Nokia N900's modem. In the end the naming is not a big problem, though, since the *ssi-protocol* only works with **PhoNet** as layer 3 protocol.

Currently there are three known userspace implementations making use of the **PhoNet** network interface:

1. Nokia Cellular Services Daemon (shipped with Maemo)

<sup>2</sup>Ubuntu modified *network-manager*, adding support for *ofono*. There are currently no plans to merge this into the upstream project, though.

## 5. Userland

2. ofono
3. fsogsmd / libgisi

All implementations expose the modem functionality via a [D-Bus](#) interface providing a more or less high-level [API](#). The daemon shipped with Maemo is the most tested application and comes with advanced PulseAudio plugins, which handle the speech data and implement noise cancellation on the OMAP's [Digital Signal Processor \(DSP\)](#). Neither the cellular services daemon, nor its accompanying PulseAudio plugins are free software, though.

In collaboration with Intel and some other vendors, Nokia started to work on a free phone daemon called ofono in 2009 with the intention to use it for MeeGo in the future. The daemon is maintained under the `kernel.org` umbrella and supports most of the N900 modem's functionalities. Not implemented is support for handling the `cmt-speech` device, so voice calls are not working without further software support. For this task a PulseAudio module has been written, which listens for call related events from ofono and properly post-processes the `cmt-speech` data using PulseAudio.

Apart from support in ofono there is also support in the [GSM](#) daemon from the [FSO](#) project (`fsogsmd`), which stripped some code parts of the ofono project and moved them into a library. This library implements a C [API](#) for the [ISI](#) protocol, which is a datagram based protocol used on top of the [PhoNet](#) protocol. It is used instead of the AT style commands, that are specified in ETSI GSM 07.07 and usually used for controlling a GSM modem.

The [ISI](#) protocol is capable of exchanging asynchronous events, for example to notify any incoming calls, as well as a request-response model. The protocol itself does not only provide usual [GSM](#) related commands, but also some unusual commands, e.g. for Location lookup using [Global Positioning System \(GPS\)](#). [Appendix B](#) provides a more detailed description of the protocol.

None of the three [ISI](#) protocol implementations provide a [GUI](#), which could be used to dial out or send a short message, though. Instead they provide a [D-Bus](#) interface, which is supposed to be used by [GUIs](#).

For [FSO's GSM](#) daemon there is currently one client providing access to most of its functionalities. It has been created for the [SHR](#) distribution and is called *phoneuid*. It provides a dialer, a message-, as well as a contacts-application. All of them will be „pre-started“ when the *phoneuid* application is started with the actual windows being hidden. For example clicking the desktop icon for the dialing application will instruct *phoneuid* to show the dialer window instead of initialising the dialer window. This design has been chosen to avoid slow access to the system's persistent memory when a call comes in. Without this feature, it could happen, that the [GUI](#) opened too slow to accept an incoming call on [OpenMoko's FreeRunner](#).

Nokia's developers chose a different approach in Maemo. Instead of writing a [GUI](#) for their Cellular Services Daemon, they made use of [Telepathy](#), which is a framework for instant messaging (like XMPP, ICQ or MSN) and Voice over IP (e.g. SIP). The Nokia N900 comes with a [Telepathy](#) backend, which provides [GSM/UMTS](#) capabilities. Thus a [GSM](#) call can be handled by the same application, which handles e.g. a SIP call and the application taking care of instant messaging can also take care of short messages.

This concept has been taken over for ofono. MeeGo and some of its successors use a [Telepathy](#) backend named *telepathy-ring*. Also following this way is the software stack used for Ubuntu Phone, which created their own [Telepathy](#) backend named *telepathy-ofono*, though. Currently Debian does not fully support any of the stacks on the N900. From the [FSO](#) stack only the `fso-audio` daemon is missing, which would take care of handling the `cmtspeech` interface. For the ofono stack the *telepathy-ring* plugin needs an update for

the new ofono API and the PulseAudio plugin must be packaged. The Ubuntu phone stack on the other hand is missing completely.

Unfortunately the work on [FSO](#) has mostly stalled with only a couple of fixes being applied in their repositories during the last year. While the Ubuntu phone stack is actively being worked on, it involves lots of different components and has no explicit support for the Nokia N900. Porting it to another distribution and adding support for a non Android based kernel will take more time, than available in this thesis. Thus the MeeGo software stack is the first choice regarding userland software. Instead of porting the complete mobile [GUI](#) (dialer, messaging app, ...) another Telepathy client, like Gnome's empathy can be used.

A simplified version of MeeGo's PulseAudio modules ported to PulseAudio 6.0 can be found on the DVD accompanying this thesis including the packaging files needed for Debian. Unfortunately packaging the modules required changes to Debian's PulseAudio package, since the project does not support external modules. For this reason the packaged modules cannot be uploaded to the Debian repository. The [FSO](#) project does not use PulseAudio and thus has an alternative solution ready. They created a small daemon named `fsoaudiod`, which does the speech data processing using `libcmtspeechdata` and the soundcard's [ALSA](#) interface. It does not listen to the ofono call events though. The patch in [Listing 5.3](#) adds optional support to use the daemon with ofono instead of [FSO](#)'s audio daemon.

Listing 5.3: Patch adding ofono support to the `cmt-speech` plugin of the `FSO`-audio daemon

```

1 Description: Add support for using the cmtspeechdata plugin with ofono
2 With this patch the FSO audio daemon can be used in
3 conjunction with ofono instead of the FSO gsm daemon.
4
5 diff --git a/src/plugins/gsmvoice_alsa_cmtspeechdata/Makefile.am b/src/plugins/
6   gsmvoice_alsa_cmtspeechdata/Makefile.am
7 index 688c909..03b2532 100644
8 --- a/src/plugins/gsmvoice_alsa_cmtspeechdata/Makefile.am
9 +++ b/src/plugins/gsmvoice_alsa_cmtspeechdata/Makefile.am
10 @@ -13,6 +13,7 @@ modlibexec_LTLIBRARIES = gsmvoice_alsa_cmtspeechdata_la
11 gsmvoice_alsa_cmtspeechdata_la_SOURCES = \
12   plugin.vala \
13   cmthandler.vala \
14 + ofono.vala \
15   $(NULL)
16 gsmvoice_alsa_cmtspeechdata_la_VALAFLAGS = \
17   --basedir $(top_srcdir) \
18 diff --git a/src/plugins/gsmvoice_alsa_cmtspeechdata/ofono.vala b/src/plugins/
19   gsmvoice_alsa_cmtspeechdata/ofono.vala
20 new file mode 100644
21 index 0000000..43a78cb
22 --- /dev/null
23 +++ b/src/plugins/gsmvoice_alsa_cmtspeechdata/ofono.vala
24 @@ -0,0 +1,7 @@
25 +namespace Ofono {
26 + [DBus (name = "org.ofono.AudioSettings", timeout = 120000)]
27 + public interface AudioSettings : GLib.Object {
28 +   [DBus (name = "PropertyChanged")]
29 +   public signal void property_changed (string name, GLib.Variant val);
30 + }
31 +}
32 diff --git a/src/plugins/gsmvoice_alsa_cmtspeechdata/plugin.vala b/src/plugins/
33   gsmvoice_alsa_cmtspeechdata/plugin.vala
34 index b66cf63..60e62cf 100644
35 --- a/src/plugins/gsmvoice_alsa_cmtspeechdata/plugin.vala
36 +++ b/src/plugins/gsmvoice_alsa_cmtspeechdata/plugin.vala
37 @@ -28,6 +28,7 @@ class FsoAudio.GsmVoiceCmtspeechdata.Plugin : FsoFramework.AbstractObject
38   private FsoFramework.Subsystem subsystem;
39   private CmtHandler cmthandler;
40   private FreeSmartphone.GSM.Call gsmcallproxy;
41 + private Ofono.AudioSettings ofonoaudiosettings;
42
43 //

```

## 5. Userland

```
41 // Private API
42 @@ -52,6 +53,12 @@ class FsoAudio.GsmVoiceCmtspeechdata.Plugin : FsoFramework.
    AbstractObject
43     }
44 }
45
46 + private void onOfonoAudioSettingsPropertyChanged( string name, Variant val )
47 + {
48 +     if (name == "Active")
49 +         cmthandler.setAudioStatus( val.get_boolean() );
50 + }
51 +
52 //
53 // Public API
54 //
55 @@ -60,14 +67,22 @@ class FsoAudio.GsmVoiceCmtspeechdata.Plugin : FsoFramework.
    AbstractObject
56     this.subsystem = subsystem;
57     cmthandler = new CmtHandler();
58
59 +     var ofono = config.boolValue( MODULE_NAME, "hook_ofono", false );
60 +
61     try
62     {
63 -         gsmcallproxy = Bus.get_proxy_sync<FreeSmartphone.GSM.Call>( BusType.SYSTEM, "
org.freesmartphone.ogsmd", "/org/freesmartphone/GSM/Device", DBusProxyFlags.
DO_NOT_AUTO_START );
64 -         gsmcallproxy.call_status.connect( onCallStatusSignal );
65 +         if (ofono) {
66 +             ofonoaudiosettings = Bus.get_proxy_sync<Ofono.AudioSettings>( BusType.
SYSTEM, "org.ofono", "/n900_0", DBusProxyFlags.DO_NOT_AUTO_START );
67 +             ofonoaudiosettings.property_changed.connect(
onOfonoAudioSettingsPropertyChanged );
68 +         } else {
69 +             gsmcallproxy = Bus.get_proxy_sync<FreeSmartphone.GSM.Call>( BusType.SYSTEM
, "org.freesmartphone.ogsmd", "/org/freesmartphone/GSM/Device", DBusProxyFlags.
DO_NOT_AUTO_START );
70 +             gsmcallproxy.call_status.connect( onCallStatusSignal );
71 +         }
72     }
73     catch ( Error e )
74     {
75 -         logger.error( @"Could not hook to fsogsmd: $(e.message)" );
76 +         var hookeddaemon = ofono ? "ofono" : "fsogsmd";
77 +         logger.error( @"Could not hook to $hookeddaemon: $(e.message)" );
78     }
79 }
80
81 --- a/conf/nokia_n900/fsoaudioid.conf
82 +++ b/conf/nokia_n900/fsoaudioid.conf
83 @@ -16,4 +16,7 @@
84 # No settings yet
85
86 [fsoaudio.gsmvoice_alsa_cmtspeechdata]
87 -# No settings yet
88 +# The following option should be set to true if
89 +# the FSO Audio Daemon should be used with ofono
90 +# instead of the FSO GSM Daemon.
91 +hook_ofono = true
```

Once the MeeGo PulseAudio modules are integrated into the main PulseAudio project, or the project reconsiders its decision regarding external modules, the Debian packaging should be straight forward, since its dependency libcmtspeechdata is already available.

With ofono, the patched fso-audioid and telepathy running (ofono and Telepathy have already been available in Debian), it should be possible to establish voice calls (and text messaging) using empathy. This will be assumed as core smartphone functionality, so that the next chapter will focus on testing the use cases.

## 6. Function tests

In this chapter it will be analyzed if the problems mentioned in the introduction have been solved and if the main features of a smartphone are supported.

### 6.1. Updates & Security Updates

In Debian updates, as well as security updates are done using `apt-get`. Assuming, that `apt` is configured correctly, it can be done as simple as running `apt-get update` to update the package index followed by `apt-get upgrade` to install new versions of all installed packages. Alternatively a front end like `gnome-packagekit` can be used, which will automatically update the packet index regularly and notify about any updates.

In opposit to Android, Debian and other *standard Linux distributions* do not come with a fine-grained rights management, so its not possible to easily remove or grant rights to an application. On the other hand the Distribution's equivalent to Google's Play-Store does not allow any application developer access. Software contained in the distribution's archives follows certain guidelines, so that malicious software, which missuses information gained from the system is not supposed to be in there.

### 6.2. Voice Calls

The ofono [D-Bus API](#) can be used to start voice calls. This can be done using any generic D-Bus client, such as `mdbus`<sup>1</sup> as shown in [Listing 6.1](#). The modem's voice data is automatically routed to the N900's soundcard by `fso-audiod` once the call has been established.

Listing 6.1: Initialize a voice call from command line using `mdbus2`

```
1 # 1. power on modem
2 mdbus2 -s org.ofono /n900_0 org.ofono.Modem.SetProperty Powered true
3 # 2. unlock SIM by providing PIN
4 mdbus2 -s org.ofono /n900_0 org.ofono.SimManager.EnterPin Pin 1234
5 # 3. connect to network
6 mdbus2 -s org.ofono /n900_0 org.ofono.Modem.SetProperty Online true
7 # 4. call 123456789 without hiding own identity
8 mdbus2 -s org.ofono /n900_0 org.ofono.VoiceCallManager.Dial 123456789 false
```

Testing revealed, that the more abstract [Telepathy API](#) mentioned in [chapter 5](#) cannot be used easily in Debian, since both clients in its repository (`Empathy` and `kde-telepathy`) require support for 3D acceleration. Instead a proof of concept user interface (`ofono-dialer`) has been implemented for this thesis, which can be found in [Appendix F](#). The Python script requests a modem power up when started, followed by a check if the [Subscriber Identity Module \(SIM\)](#) card has enabled PIN protection. If no PIN is required, or the user

---

<sup>1</sup><https://github.com/freesmartphone/mdbus/>

## 6. Function tests

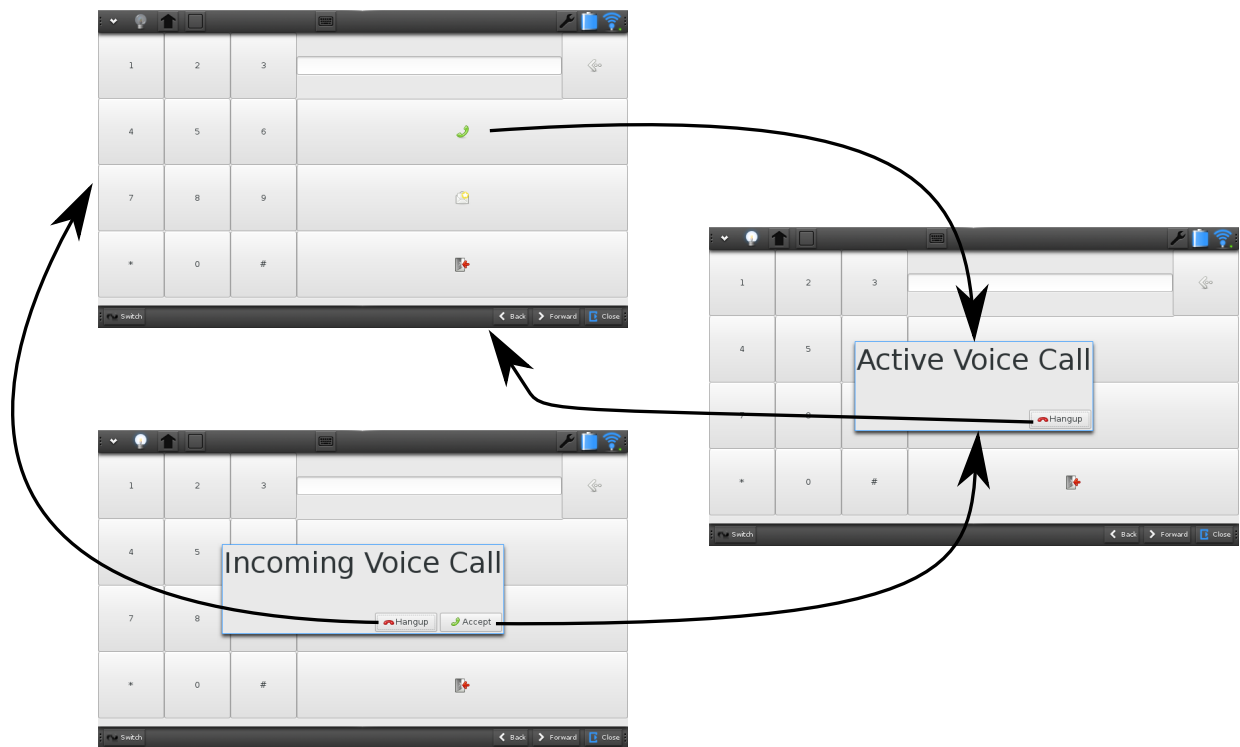


Figure 6.1.: Voice call handling in ofono-dialer

provided the correct one in the following PIN dialog, the script will connect the modem to the network and open the top-left window seen in [Figure 6.1](#). In this window the to-be-called phone number can be inserted into the input box using the phone's hardware keyboard or the virtual keypad on the left. Pressing the call button will open a dialog as visible in the right window in [Figure 6.1](#). The application also handles incoming voice calls by opening the dialog visible at the bottom. While both, outgoing and incoming calls can be established with this application, it is not very user friendly, since it lacks usability features, such as

- automatically turn off display and ignore input events if phone is held to the ear
- automatically rotate the display using the accelerometer
- support ringing/vibrating on incoming call
- support address book lookup for phone numbers
- error handling

Porting Ubuntu's or MeeGo's voice call applications might help to improve voice call handling. Since both applications require many other libraries to be ported first, further evaluation on this was not possible in the scope of this thesis.

### 6.3. Short Messages

Assuming the modem is already connected to the network (see previous section), Short Messages (SMS) can be sent using ofono's D-Bus API as shown in [Listing 6.2](#).

Listing 6.2: Send a SMS using mdbus

```
1 # send "Hello World!" to 123456789
```



```
2      mdbus2 -s org.ofono /n900_0 org.ofono.MessageManager.SendMessage 123456789 "Hello
      World!"
```

The proof of concept dialer application from [Appendix F](#) can also be used to send and receive short messages using the ofono API. [Figure 6.2a](#) shows how a SMS can be sent with the interface by starting with entering the phone number at ①. After pressing the SMS button ②, a new dialog opens to insert the message's text at ③. Once everything looks fine, the message can be send using the send button ④. For incoming messages a dialog window, as seen in [Figure 6.2b](#), opens automatically showing the incoming message.

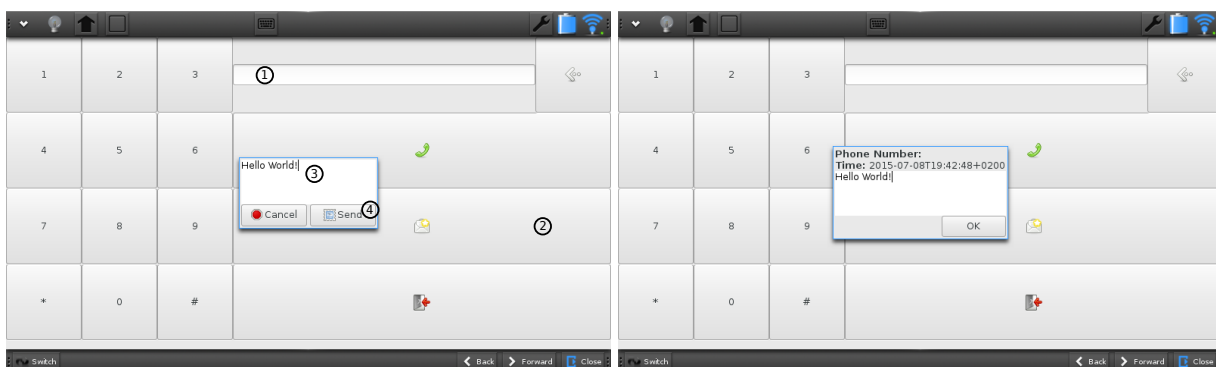
Like the voice call part, the short message implementation is just proof of concept for testing the lower stack. The features missing in the voice call part also affect the short message implementation of the application (e.g. no phone book, no history). Again porting Ubuntu's or MeeGo's short message applications might help to make the distribution ready for daily use.

## 6.4. Web Surfing

Connecting to a wireless LAN can be simply established using the *connman* configuration tool from enlightenment. Unfortunately Debian's enlightenment desktop environment is currently built without *connman* support, so the package must be rebuilt with the correct options added.

Connecting to the GPRS or UMTS network is currently not possible using enlightenments *connman* configuration tool, but can be achieved using the *connmanctl* command line tool.

Once a connection has been built up, any browser available from the Debian archive (e.g. Iceweasel (Firefox) or Gnome's epiphany) can be used the same way as on a desktop. For this thesis *midori* has been tested as visible in [Figure 5.1](#) and [Figure 6.3](#). Unfortunately the Nokia N900's system memory is too small for comfortable surfing, though. Scrolling webpages up and down will result in short intervals to load the next chunk of data into the memory.



(a) send SMS

(b) receive SMS

Figure 6.2.: SMS handling in ofono-dialer

## 6.5. Customization

There are various ways to customize the interface. One of the first relevant visible components is the display manager. Since the Nokia N900 comes with a hardware keyboard, any non [OpenGL](#) using display manager can be used. This might be a bigger issue on other smartphones, which require a virtual keyboard to insert the user's credentials. Even if *lightdm* is used there are a lot of customization options, since it comes with different user interfaces (GTK+ or Qt based) and many configuration options in */etc/lightdm*.

The next option of customization is the selection of a desktop environment. For this thesis Enlightenment's window manager has been chosen, since it comes with a nice mobile optimized variant. Of course it is possible to use any other window manager instead (as long as it does not require [OpenGL](#), which is currently not supported by the N900). Basically everything can be adopted to fit the users needs, only limited by the amount of time the user is willing to dedicate. For example [Figure 6.3](#) shows LXDE running on the Nokia N900 resulting in a PC like handling of the smartphone including a start menu in the bottom left corner.

If Debian does not provide the desired functionality in one of its 43000 packages, the system is also flexible regarding to the development of custom applications. It is possible to use most programming languages, frameworks and graphical toolkits on the Nokia N900. For example the dialer for testing the voice call and short message service has been written in Python using GTK+. If software is not written in an interpreted language, it must be compiled for usage with an ARM processor, though. This can either be done by using a cross-compiler, or directly on target by installing the compiler on the phone.

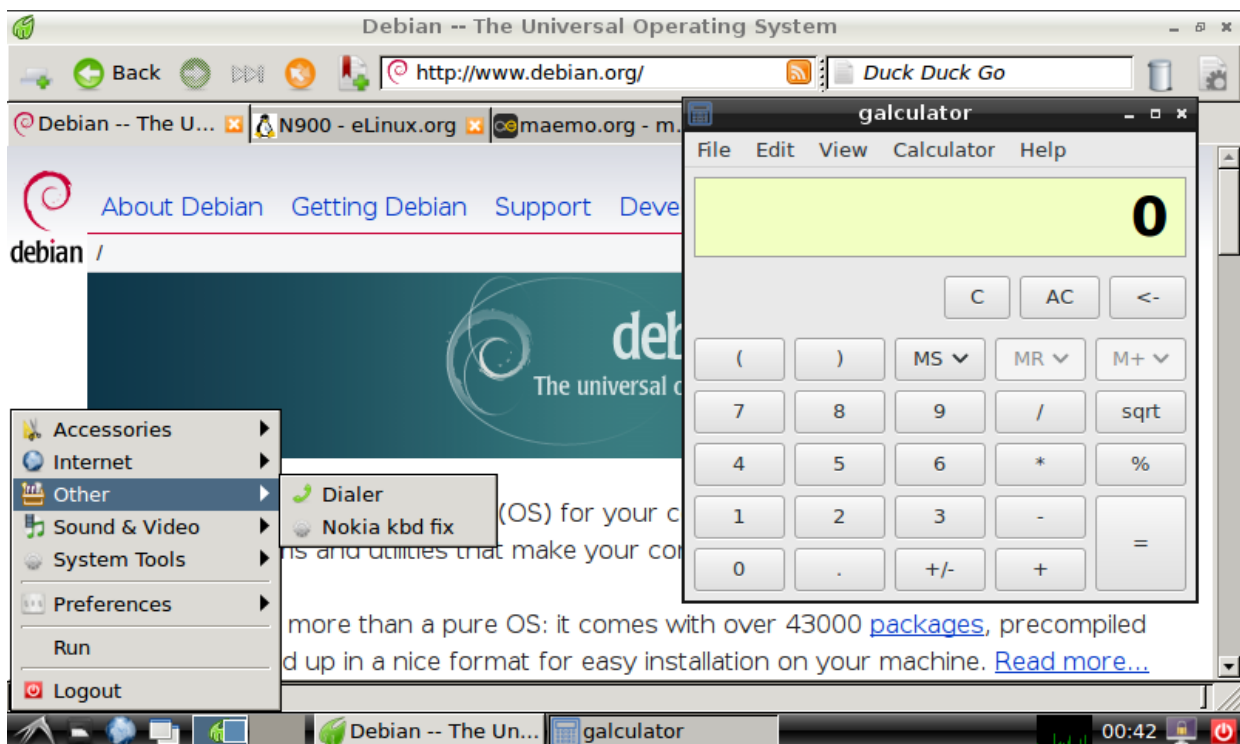


Figure 6.3.: LXDE running on the Nokia N900

## 7. Outlook

This chapter aims to give a summary of outstanding work to improve the usability of the N900 running a mainline kernel and documents some missing smartphone related applications in Debian. Initially it will have a look at improvements for a next revision of the debug adapter described in [chapter 3](#) though.

### Debug Adapter

As already mentioned in [chapter 3](#), the debug adapter should be capable of cutting the phone's power supply, so that it can be used for automatic regression testing. In this scenario there is often no local user available, that can do this manually in case of a hanging kernel. The adapter built in [chapter 3](#) provides this feature, but due to a hardware bug this feature is not usable. A future revision of the adapter should replace the transistor used for cutting off the power with a relay, which supports cutting the power independently of the voltage direction (which depends on the battery charging state).

Apart from that it would be nice if the adapter could optionally supply the phone using the power from its [USB](#) port instead of requiring a battery. It must not be used exclusively, though, since it would make it impossible to test the battery measurement and charging logic. This feature requires prior measurements of the maximum current drawn by the N900 to ensure, that it does not damage any [USB](#) ports.

The adapter's next revision should also use another connector for attaching the battery compartment, since the current one provides only a few mating cycles and is quite fragile. A suitable replacement would be a standard 2.54 mm pin header.

Another improvement would be the usage of shorter [Pogo-Pins](#). The adapter built in this thesis only uses the long [Pogo-Pins](#), since no other pins have been found, which satisfy all requirements. The replacement pins should have approximately the same diameter and a round head to avoid damage to the test pads, but should be much shorter to reduce the height of the adapter.

Apart from that it has been noticed, that the [PCB](#) should provide more of its functionality (like the [I2C](#) port) on externally available pins, so that debugging is eased and extra functionality may be attached at a later time. Also useful for debugging purposes are vertical interconnect accesses commonly known as via. These make signal probing using an oscilloscope or just a multimeter for testing, if everything has been soldered correctly, much easier.

Last but not least the [QFN32](#) packaged [USB-UART](#) chip should be replaced by another chip, which uses a package with pins instead of pads (e.g. FTDI's FT2232D, which is available in a LQFP-48 package). While the chip has been soldered successfully using hot-air, its placement turned out to be very hard without a pick-and-place machine, since one cannot see if the pads of the chip are aligned with the [PCB](#)'s pads while placing it.

Alternatively to the dual [UART](#) chip a [USB](#) hub should also be considered, since the additional [USB](#) port

## 7. Outlook

could be attached to the N900's [USB](#) port, so that a single [USB](#) cable can be used to attach the development phone to the development PC. To reduce the required area on the [PCB](#), a chip with integrated [I2C](#) support can be used as described in Microchips application note 1941 [25].

The next sections will have give an outlook regarding the Linux kernel support for specific hardware components of the Nokia N900 starting with its modem support.

### Modem

As described before the modem's power up/down sequence is currently handled by userland code via exported [GPIOs](#). This is supposed to be moved into kernelspace, so that userspace software like ofono can simply open the [PhoNet](#) network device and focus on the [ISI](#) implementation. All power sequence related steps should be done inside of the kernel.

Apart from that there are concepts to move the speech handling into the kernel. This would make it possible to do voice calls without any special userspace applications (except for ofono / fso-gsm). Audio data could simply be piped from the modem's [ALSA](#)-device to the sound card's [ALSA](#)-device and vice versa. This task requires a complete rework of the cmt-speech driver, which would have to react to call related packets sent through the related [PhoNet](#) device. In addition to the new code detecting speech data related events on the [PhoNet](#) device, the code from the cmtspeech PulseAudio module / fso-audiod and libcmtspeechdata must be moved inside the cmt-speech kernel driver.

### BCM2048

As described in [section 4.5](#) a clean Bluetooth driver for serial attached devices from Nokia is almost ready. Before sending the driver attached to this thesis for mainline inclusion, the communication bug has to be solved first, though. This requires to debug the differences between the new driver and Nokia's old driver regarding to serial port configuration and GPIO timings.

In addition the firmware required for the BCM2048 chip is not available under a license compatible with the linux-firmware project. This makes installing a standard Distribution on the Nokia N900 a little harder than it has to be, since the firmware must be extracted from the original Operating System. This problem can be solved by negotiating a more fitting license with its current copyright holders (Microsoft).

Also waiting for a few cleanups is the BCM2048 [FM](#) radio driver, which is currently part of the *staging* tree inside of the mainline kernel. If nobody takes care of the requested cleanups, it will be removed from the mainline kernel in the next few releases.

### Camera Subsystem

Unfortunately one of the tasks, which accompanied this thesis could not be finished in time with it, so that the camera subsystem is still not working at all with the mainline kernel.

As previously described in [section 4.6](#), time intensive debugging sessions are needed to figure out communication problems between the camera sensors and the [ISP](#). In addition the driver for the main camera sensors (ET8EK8) and the related driver for its auto-focus motor have not yet been cleaned up for mainline inclusion and are missing proper [DT](#) bindings.

Apart from that some changes are required to the asynchronous [V4L2 API](#) are needed to support sensors, which are not directly connected to an [ISP](#).

## PowerVR SGX (GPU)

Last but not least the smartphone's [GPU](#), which takes care of 2D and 3D acceleration is still a complete black box. While it has been built into multiple smartphones and some of Intel's [SoCs](#), reverse engineering the [IP-Core](#) is still in its very early phases with nobody actively working on it.

Since more and more modern desktop environments available for Linux use 3D acceleration by default, this will be a task, which has to be tackled at some point to keep the Nokia N900 or its unofficial successor the Neo900 (which also uses the PowerVR SGX) an usable platform.

While there [Texas Instruments](#) released an [GNU Public License \(GPL\)](#) licensed kernel driver for the PowerVR SGX, it is considered non-free as long as it depends on a non-free userspace component. This is especially true, when the kernel component is mostly used to forward data between the userspace component and the actual [IP-Core](#), which is the case for the PowerVR SGX.

Nevertheless the existing software is a good starting point to reverse engineer the [GPU](#), since all data being exchanged with the PowerVR SGX can simply be tapped in the kernel.

For proper support in common [Linux distributions](#), multiple components would be required. First of all there would have to be a kernel driver, which takes care of providing the userspace components an interface to the [GPU](#) similar to the existing driver. Except for initial testing it does not make sense to reuse [Texas Instruments](#)'s driver, which abstracts the Linux kernel [APIs](#) (possibly to keep the PowerVR SGX driver compatible with other operating systems). This driver would have to be integrated with the existing omapdrm driver.

Next support for the driver specific interfaces should be added to libdrm, which provides an [API](#) to the rendering related interfaces provided by the kernel.

Last but not least the main task is the creation of a Gallium3D based userspace driver, which interacts with the kernel driver using the libdrm. For this task the PowerVR reverse engineering project has already analysed the instruction set used for the [unified scalable shader engine \(USSE\)](#) modules found inside of the PowerVR SGX [19].

According to one of Imagination Technology's employee, the vendor is currently working on a free driver for the PowerVR series [2]. Apart from this hint and a matching job advertisement, no details have been made public until the end of this thesis, though. In the past these kind of in-house developed drivers had problems to be merged into the official projects (e.g. the VIA drivers never have been merged). Nevertheless it may help to give some insights about the hardware's functionality.

## 7. Outlook

Apart from the hardware support deficiencies, there are also some smartphone related applications missing. These will be described in the next section.

### **Userspace Software**

Nowadays users of Android, iOS or Windows Phone based smartphones have access to a wide array of mobile optimized applications. While some of them have an equivalent in the currently common Linux distributions, many do not. While it is possible to use some of the applications written for normal personal computers and notebooks, they are often not optimized for usage with a touchscreen. Having easily usable user interfaces is important at least for the core applications, such as a dialer, a messaging application or a calendar. With the existing infrastructure built up in this thesis it should be possible to port the existing applications from the Maemo and MeeGo project, write new ones or package the Ubuntu Phone stack.

The next chapter will give a conclusion regarding running a universal Linux distribution on a smartphone.

## 8. Conclusion

While there is lots of outstanding work to make a non-mobile optimized [Linux distribution](#), such as Debian, properly usable on a smartphone, it is possible to do voice calls and exchange text messages. Due to missing Touchscreen optimized software the workflow required for doing different smartphone tasks is less straight forward on a universal [Linux distribution](#), though. Here the N900's hardware keyboard helps a lot, since it is mostly possible to use the phone like a notebook.

Even with proper alternatives available for the most common smartphone tasks, the Debian is probably not usable on a smartphone for the average user, though. Since the distribution's software repositories contain many software components not intended to be used on smartphones while missing typical applications used by casual users (e.g. games) third party repositories would be required. This is a security problem, though, since there is no fine-grained rights management available.

Nevertheless, the distribution is interesting for students and open source developers, since they can simply reuse their existing software on a phone. For people trusting the [GNU is not unix \(GNU\)](#) userland and the [Linux](#) kernel it is also a security benefit, as no closed source bits are needed and the software can be updated without being dependent on any specific vendor.

Unfortunately only the Nokia N900 is almost fully supported by the mainline [Linux](#) kernel. As visible in [chapter 3](#) and [chapter 4](#) adding support for a smartphone is a time consuming and demanding task if its not done large-scale (e.g. directly by the vendor). Analysis why this is not yet happening has been done by the Consumer Electronics Linux Forum revealing that 91% of the developers think its important, but in 61% the management disapproves of such work or does not grant the required time [8].

One of the biggest issues regarding userland software is the N900's missing support for 3D acceleration and its low amount of memory (256MB), which is barely enough to start the base system. Modern browsers require a multiple of that and keeping some memory free helps the [Linux](#) kernel to cache some files. It is expected, that a modern smartphone, which usually comes with at least one GB of memory, will solve this problem.

## Appendix



## A. Debian Image

There are multiple ways to create a N900 compatible Debian image. This chapter will use *debootstrap*, which can be found in the Debian archive:

```
https://packages.debian.org/stable/debootstrap
```

This tool allows to install a Debian userland into an existing directory in a filesystem. It has intentionally quite few dependencies, so that it could be used on a non-Debian based system by unpacking the \*.deb archive. Once *debootstrap* is available on the system the following steps will create a Debian root filesystem.

### Preparing the SD card

After plugging the  $\mu$ SD card into your computer's card-reader, it will appear as */dev/sd?* or */dev/mmcblk?* depending on the card-reader type. In the following \$SDCARD is used as placeholder for this device.

Once the correct device has been identified it should be partitioned for use with the N900. It is recommendable to create a small FAT32 partion to be used with U-Boot (The U-Boot release in Maemo does not support ext2) and a second partition for the root filesystem. This can be done by using parted or its graphical interface gparted. In the following \$BOOTPART will be used to represent the (unmounted) boot partition and \$ROOTPART will be used to represent the (unmounted) root partition.

Once this has been done the root partition should be mounted at some point in the filesystem. In this chapter we will expect, that \$ROOTPART is mounted to */mnt/n900*. This can be done by running

```
1 mkdir /mnt/n900
2 mount $ROOTPART /mnt/n900
```

### Running debootstrap

With the filesystem prepared and debootstrap installed the installation process can be started. A standard root filesystem can be created by using the following command.

```
1 debootstrap --arch=armhf --foreign stable /mnt/n900 http://httpredir.
   debian.org/debian
```

It is possible to include some desired packets at installation time though, so it is recommendable to extend the query to the following and use testing instead of stable to get a kernel with the ARM Errata 430973 fix.

```
3 export PACKAGES="ifupdown,openssh-server,udev,procps,netbase,vim"
4 export PACKAGES="$PACKAGES,module-init-tools,wget,openssh-client"
```

## A. Debian Image

```
5 export PACKAGES="$PACKAGES,watchdog,whiptail,alsa-base,apt-utils"
6 export PACKAGES="$PACKAGES,locales,dbus,psmisc,htop"
7 export PACKAGES="$PACKAGES,tmux,ofono,alsa-utils,mdbus2,rsyslog"
8 export PACKAGES="$PACKAGES,console-setup,aptitude,u-boot-tools"
9 export PACKAGES="$PACKAGES,net-tools,wireless-tools,rfkill,wpa_supplicant"
10 export PACKAGES="$PACKAGES,iputils-ping,isc-dhcp-client"
11 export PACKAGES="$PACKAGES,linux-image-armmp"
12 debootstrap --arch=armhf --foreign --verbose --variant minbase --include=
    $PACKAGES testing /mnt/n900 http://httpredir.debian.org/debian
```

Once this has been successfully completed there will be an armhf root filesystem in */mnt/n900*. Since *--foreign* was supplied the packages have not yet been configured, though. For this task either a working ARMv7 board or an emulator is required. The root filesystem itself is not yet bootable, since the kernel's *initramfs* has not yet been created.

The easiest way to configure the packages is using a *chroot* and *qemu* with userland emulation in combination with an entry in the PC's Linux kernel *binfmt\_misc* interface. This way ARM binaries can be called on the PC resulting in the kernel running the binary through *qemu-user*. In Debian the following listing will change into the N900's new root filesystem (the *binfmt\_misc* entry is created automatically by installing *qemu-user-static*):

```
1 apt-get install qemu-user-static
2 cp /usr/bin/qemu-arm-static /mnt/n900-rootfs/usr/bin
3 chroot /mnt/n900-rootfs /bin/bash
```

Once a shell in the N900 root filesystem is available, the post configuration can be started. *debootstrap* will put a script into the root filesystem at */debootstrap/debootstrap*, which should be called with *--second-stage*:

```
1 /debootstrap/debootstrap --second-stage
```

This will configure all packages downloaded and extracted by *debootstrap* in the previous step. Next some extra configuration steps are required, which are normally performed by the Debian installer.

## Initial Configuration

Once all installed packages have been configured the root filesystem could be used in theory. In practice a couple of additional configuration steps should be done. For this step we use the N900 root filesystem shell already opened using *chroot* previously. First of all a password for the root account should be set using *passwd*:

```
1 passwd
```

Then the keyboard should be configured by editing */etc/default/keyboard* using an installed editor (e.g. *vim*). The file should set the following variables for the built-in keyboard of the smartphone.

```
1 XKBMODEL="nokiארx51"
2 XKBLAYOUT="de"
```

Next the hostname should be configured to setup */etc/hostname* to avoid accidentally working in the wrong filesystem.

```
1 echo "HOSTNAME" > /etc/hostname
```

Another step, which should be taken is disabling a *udev* rule shipped with Debian, which takes care of persistent network device names. This rule normally ensures, that a WLAN card will always get the same device name (e.g. wlan0). Unfortunately this rule has the opposite effect on the N900, since the WLAN card has no static MAC address configured. Instead it generates a random one at each boot resulting in each boot having another WLAN device name. The rule can be disabled by deleting or commenting the related file in */etc/udev/rules.d*.

```
1 echo "#" > /etc/udev/rules.d/75-persistent-net-generator.rules
```

Next the setup for apt sources should follow. Note that non-free is only included as source for the WLAN firmware. If WLAN firmware is not required (wl1251 won't work without the firmware!) or not sourced using the Debian infrastructure the non-free entry can be skipped.

```
1 echo "deb http://httpredir.debian.org/debian testing main" > /etc/apt/sources.list
2 echo "deb http://httpredir.debian.org/debian testing non-free" >> /etc/apt/sources.list
3 apt-get update
```

Last but not least a user account should be prepared, so that the device does not have to be used with the root account. This can be done using the *adduser* command.

```
1 adduser <username>
2 adduser <username> audio
3 adduser <username> video
4 adduser <username> bluetooth
```

## Hardware Specific Configuration

Next some of the hardware components need additional configurations to make it usable or improve its usability. First of all there is the WLAN chip, which requires a firmware file. Installing *firmware-ti-connectivity* is enough to get the WLAN device working.

```
1 apt-get install firmware-ti-connectivity
```

Next the modem driver currently does not automatically load in all cases, so it should be added to */etc/modules*. Apart from that the *pm* parameter should be set, so that the driver exports the **GPIOs**. Last but not least the **GPIOs** are exported to */sys/bus/hsi/devices/n900-modem*, but ofono searches for them in */dev/cmt* for historical reasons. Thus */dev/cmt* should be linked to the correct location.

```
1 echo nokia-modem >> /etc/modules
2 echo "options nokia-modem pm=1" >> /etc/modprobe.d/nokia-modem.conf
3 ln -s /sys/bus/hsi/devices/n900-modem /dev/cmt
```

Apart from that the display drivers should be loaded as early as possible, since its the main source of debug outputs without a serial adapter. This requires adding a couple of modules to */etc/initramfs-tools/modules* and regenerating the *initramfs*.

## A. Debian Image

```
1 echo "spi_omap2_mcspi" >> /etc/initramfs-tools/modules
2 echo "omapdss" >> /etc/initramfs-tools/modules
3 echo "panel_sony_acx565akm" >> /etc/initramfs-tools/modules
4 echo "omapdrm" >> /etc/initramfs-tools/modules
5
6 update-initramfs -k all -u
```

At last a couple of X.org related packages should be installed to have a graphical user interface. Debian provides for example *e17* as smartphone compatible desktop environment. In addition a simple browser and a terminal emulator is installed with the following commands.

```
1 apt-get install xserver-xorg xserver-xorg-video-omap xserver-xorg-input-
  evdev xinit
2 apt-get install xinput-calibrator x11-utils xterm xinput
3 apt-get install e17
4 apt-get install terminology mpv midori
```

## Boot Partition

Next, the `$BOOTPART` must be mounted (outside of the chroot). In the thesis `/mnt/n900/bootloader` will be used as mount point.

```
13 mount $BOOTPART /mnt/n900/bootloader
```

Maemo's U-Boot package is configured to check for a *boot.scr* file on the SD cards first partition, if it is using a FAT32 filesystem. If it is found, it will be sourced by the bootloader. The following boot script can be used, assuming, that the Device Tree blob will be attached to the kernel instead of being loaded individually by the bootloader.

```
1 setenv bootcmd 'mmc init; fatload mmc1 0 0x82000000 uImage; fatload mmc1
  0 0x83000000 uInitrd; bootm 0x82000000 0x83000000'
2 setenv bootargs 'root=/dev/mmcbk0p2 rootwait console=ttyO2,115200n8
  earlyprintk loglevel=7 quiet rootwait rw'
3 boot
```

To be usable with U-Boot the boot script must be compiled first. Assuming this boot script has been named *boot.scr.txt*, the usable boot script can be generated by using *mkimage* (available in Debian's *u-boot-tools* package) as follows:

```
1 mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "Debian" -d boot.
  scr.txt boot.scr
```

The resulting *boot.scr* file should be placed at `/mnt/n900/bootloader`, so that it will be picked up by U-Boot. Additionally the kernel and the initramfs must be placed to `/mnt/n900/bootloader`. This task requires concatenating the kernel image and the Device Tree blob, which can be done by using *cat*. Apart from that the initramfs file will be copied to a temporary location for generating the U-Boot images:

```
1 cat /mnt/n900/boot/vmlinuz-4.0.0-2-armmp /mnt/n900/usr/lib/linux-image
  -4.0.0-2-armmp/omap3-n900.dtb > zImage
2 cp /mnt/n900/boot/initrd.img-4.0.0-2-armmp > initrd
```

Last but not least the files must be converted into U-Boot's image format, which can also be done using *mkimage*. The temporary files *zImage* and *initrd* can be deleted afterwards.

```
1 sudo mkimage -A arm -O linux -T kernel -C none -a 80008000 -e 80008000 -
  n vmlinuz -d zImage /mnt/n900/bootloader/uImage
2 sudo mkimage -A arm -O linux -T ramdisk -C none -a 0 -e 0 -
  n initramfs -d initrd /mnt/n900/bootloader/uInitrd
3 rm zImage initrd
```

After this the userland is ready for use on the smartphone.

## Bootloader Installation

Since the N900's bootloader, also known as *NOLO*, cannot be replaced easily, U-Boot is installed as chain-loaded bootloader. NOLO will load it the same way, as it would load a normal kernel image. Flashing the U-Boot image can be done either from a working system on the N900 (e.g. Maemo), or via NOLO's flash-over-USB feature.

**U-Boot from Debian:** If Debian's U-Boot image should be used, it is not easily possible to keep using Maemo on the device. The basic workflow consists of installing the package *u-boot-omap* package in the N900's root filesystem, which provides pre-built U-Boot images for the smartphone.

```
1 apt-get install u-boot-omap
```

Next a tool to access the N900's first bootloader (NOLO) must be installed in the host system. For this step either Nokia's closed source tool *flasher-3.5* can be used, or the reverse engineered *0xFFFF* (Free Fiasco Firmware Flasher). In Debian *0xFFFF* is available from the main repositories, so it will be used here.

Before flashing the U-Boot image, it can be directly started for testing purposes.

```
1 0xFFFF -l -m kernel:/mnt/n900/usr/lib/u-boot/nokia_rx51/u-boot.bin
```

Once its known to be working the *-l* can be skipped to flash the U-Boot image to the kernel area.

**U-Boot from Maemo:** If, on the other hand, the existing Maemo installation should be kept easily available, U-Boot can be installed from within Maemo. In this case the package will append Maemo's own kernel to the U-Boot image, so that it is still available.

Assuming, that the user has enabled the community repositories in Maemo installing U-Boot is as easy as running the following command from within Maemo:

```
1 apt-get install uboot-pr13
```

If the U-Boot Script has been configured as described in the previous section, the bootloader will prompt for operating system selection if the keyboard has been slid out during the boot process. Otherwise Maemo will be booted by default.

## B. ISI-Protocol

ISI is a datagram based protocol used for communication with Nokia phones. Every ISI packet starts with a common header visible in [Table B.1](#), which is identical to the [PhoNet](#) protocol.

Offset	Type	Name
0x00	UINT8	Receiver Device ID
0x01	UINT8	Sender Device ID
0x02	UINT8	Resource ID
0x03	UINT16	Length
0x05	UINT8	Receiver Object
0x06	UINT8	Sender Object

Table B.1.: PhoNet Header

This header contains the receiver and sender device IDs, which could be either *0x00* for the Modem, *0x6c* for the Host or *0xFF* for Broadcast. Next follows a resource identifier. A list of the most important resource IDs can be seen in [Table B.2](#). Next the header contains a length field, which contains the number of bytes following after the [PhoNet](#) header. Last but not least it contains a number describing a receiver and sender object, which can be used to identify, which process has requested specific information, in case of multiple processes communicating with the modem.

Value	Description
0x01	Call
0x02	Short Message Service (SMS)
0x06	Subscriber Services
0x08	SIM Authentication (PIN, PUK)
0x09	SIM
0x10	ComMgr
0x0A	Network
0x1B	Phone Information
0x31	GPRS
0x54	Location (GPS)
0xB4	Radio settings
0xD9	Pipe

Table B.2.: ISI Resource IDs (excerpt of most important IDs)

The [PhoNet](#) header is followed by a resource specific header, which usually starts with packet identifier followed by a command identifier.

A full list of all gathered (e.g. from ofono) and reverse engineered information about the protocol can be found in the Wireshark plugin on the attached DVD. The plugin implements a protocol dissector for the ISI protocol.

Offset	Type	Name
0x00	UINT8	Packet ID
0x01	UINT8	Common Command

Table B.3.: ISI Header

As an example, how the protocol is used, a commented recording of a [GPS session](#)<sup>1</sup> will be described in the next sub-chapter.

## GPS-Session

As example for the [ISI](#)-protocol this chapter will have a look at a typical GPS session as invoked by Nokia's map application. [Figure B.1](#) gives a rough overview over a typical GPS session, which will be analysed more detailed in the following paragraphs.

First of all the host system registers for GPS resource status indications sent by the modem. This is done using the *ComMgr* resource. The related packet can be seen in [Table B.4](#).

Offset	Key	Value	Description
0x00	Receiver Device	0x00	Modem
0x01	Sender Device	0x6c	Host
0x02	Resource	0x10	ComMgr
0x03	Length	0x0006	(0x5+0x6 = 0x0b)
0x05	Receiver Object	0x00	
0x06	Sender Object	0x62	
0x07	Packet ID	0x00	
0x08	Command	0x10	Subscribe Resources Indication
0x09	Resource Count	0x01	
0x0A	Resource ID	0x54	GPS

Table B.4.: GPS Session: Status Indication Subscription

Next the GPS receiver must be powered on. For this a Hybrid Tracking request must be sent to the modem. The related packet sent by Maemo can be seen in [Table B.5](#). It basically informs the modem, that the host systems requests hybrid location data. Thus it will try to get location data from multiple sources (e.g. GSM and GPS).

Next the modem will reply to the power request with an matching Hybrid tracking response. The related packet can be seen in [Table B.6](#).

The following packets (Socket Open, Socket Send, Socket Receive) are used by the modem to request A-GPS information. This is initiated by a Socket open request from the modem, which tells the Host system to open a TCP connection to the specified server. [Table B.7](#) contains the socket opening request sent by an unmodified Maemo system.

It is replied to by the Host system with a simple response containing a descriptor ID used for further

<sup>1</sup>The GPS receiver of the N900 is connected to the modem instead of the processor, so GPS positions are queried from the modem.

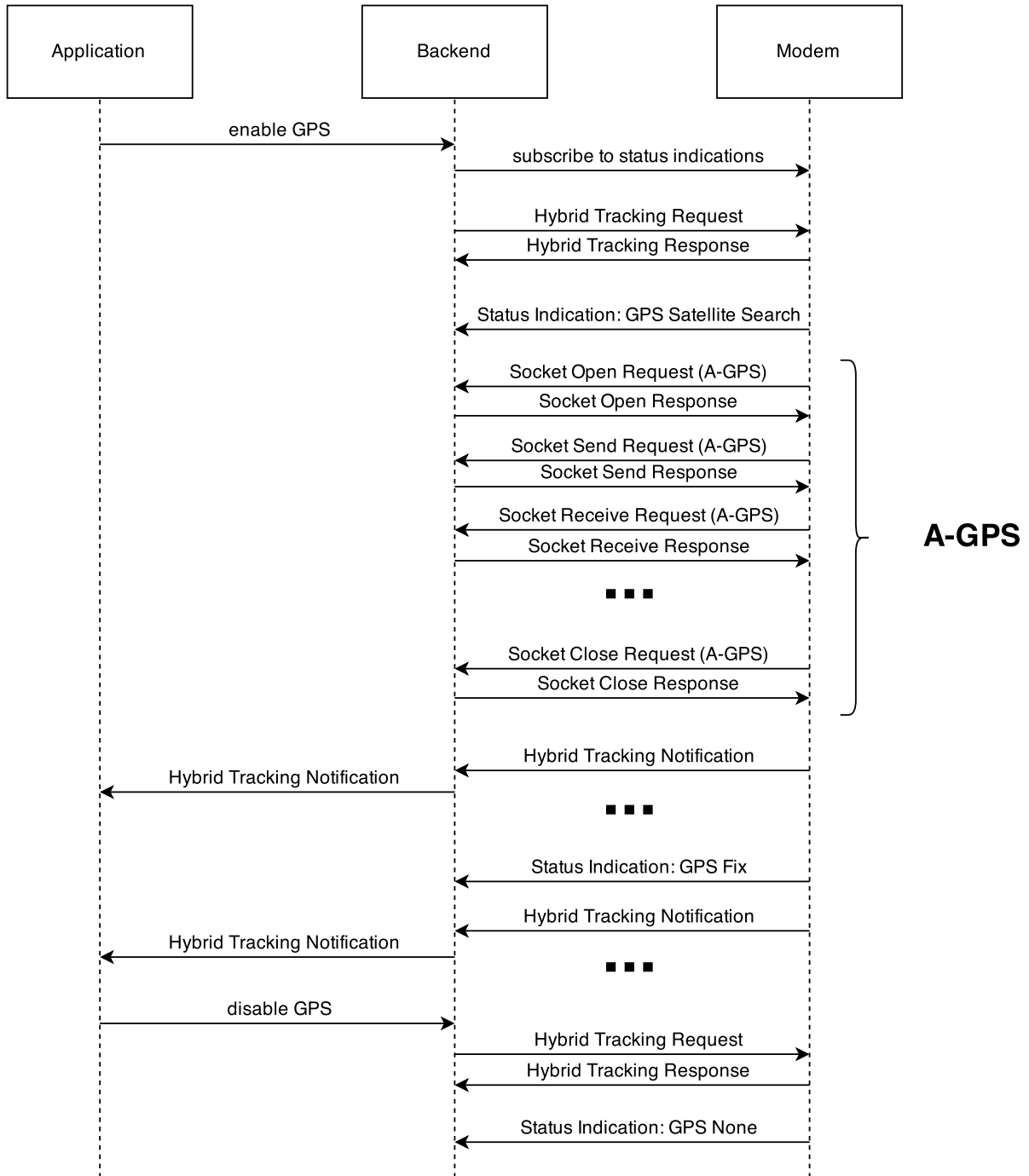


Figure B.1.: GPS Session



Offset	Key	Value	Description
0x00	Receiver Device	0x00	Modem
0x01	Sender Device	0x6c	Host
0x02	Resource	0x54	GPS
0x03	Length	0x001e	(0x5+0x1e = 0x23)
0x05	Receiver Object	0x00	
0x06	Sender Object	0x62	
0x07	Packet ID	0x00	
0x08	Command	0x90	Hybrid Tracking Request
0x09	Hybrid Command	0x00	Measurement Start
0x0a	Number of sub-packets	0x01	
0x0b	Client-ID	0x00000000	
0x0f	NPE ID	0x00000000	
0x13	sub-packet command	0x0901	Hybrid Tracking Instruction
0x15	sub-packet length	0x0010	Length
0x17	allowed methods	0x0000000a	ACWP, AGNSS
0x1b	Options	0x00000003	channel status, cell info
0x1f	Interval	0x000a	1 second
0x21	Report Criteria	0x00	Hybrid Report at all Intervals
0x22	ACWP Timer	0x00	-

Table B.5.: GPS Session: Hybrid Tracking Request

Offset	Key	Value	Description
0x00	Receiver Device	0x6c	Host
0x01	Sender Device	0x00	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x000e	(0x5+0xe = 0x13)
0x05	Receiver Object	0x62	
0x06	Sender Object	0x00	
0x07	Packet ID	0x00	
0x08	Command	0x91	Hybrid Tracking Response
0x09	Status	0x01	OK
0x0a	Cause	0x00	Cause Success
0x0b	Client ID	0x00000000	
0x0f	NPE ID	0x00000000	

Table B.6.: GPS Session: Hybrid Tracking Response

processing on this socket and a status code. Once the modem got the response, it will request ([Table B.8](#)), that some data is sent via the socket.

The Host replies with two packets to the Modem. The first packet, known as Socket-Send-Response, contains the Descriptor ID as reference together with a status code. If there were no problems it will be followed by another packet, which contains the received data as shown in [Table B.9](#).

This packet will be replied by the Modem with a matching Socket-Receive-Response. Next the Modem will either send another Socket-Send-Request or will request to close the Socket using a Socket-Close-Request answered by a Socket-Close-Response packet.

Offset	Key	Value	Description
0x00	Receiver Device	0x6c	Host
0x01	Sender Device	0x00	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x002a	(0x5+0x2a = 0x2f)
0x05	Receiver Object	0x4e	
0x06	Sender Object	0x1b	
0x07	Packet ID	0x01	
0x08	Command	0x84	Socket Open Request
0x09	Number of sub-packets	0x02	2
0x0a	Sub-packet Type	0x1200	Socket Address
0x0c	Sub-packet Length	0x001c	
0x0e	Port	0x1c6b	7275
0x10	Type	0x02	Domain Name
0x11	IP Length	0x00	
0x12	Domain Length	0x0e	
0x13	Padding	0x000000	
0x11	Data	supl.nokia.com	

Table B.7.: GPS Session: Socket Open Request

Offset	Key	Value	Description
0x00	Receiver Device	0x6c	Host
0x01	Sender Device	0x00	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x002a	(0x5+0x2a = 0x2f)
0x05	Receiver Object	0x4e	
0x06	Sender Object	0x1b	
0x07	Packet ID	0x01	
0x08	Command	0x86	Socket Send Request
0x09	Descriptor	0x09	
0x0a	Padding	0x00	
0x0b	Data Length	0x0040	
0x0d	Padding	0x0000	
0x1f	Data	...	

Table B.8.: GPS Session: Socket Send Request

As implied by the domain name and the port number in the socket opening request, the data being sent around is following the [Secure User Plane Location Protocol \(SUPL\)](#) protocol. This protocol is used to inform the user about his or her rough position using information about neighbouring [GSM](#) cell IDs (or alternatively WLAN networks). It also provides information about the current [GPS](#) satellite positions and their route, so that the [GPS](#) receiver can find them faster and does not have to receive the data using [GPS](#) satellite signalling, which is very slow. During the analysis of this, it has been noticed, that the phone's [International Mobile Subscriber Identity \(IMSI\)](#) is sent around as session ID in the [SUPL](#) connection making the phone more vulnerable to low level attacks[10].

Some time after requesting the location, the modem will send out a GPS status indication, that the GPS lock has not yet been acquired (satellites have not yet been found). The related packet is described in [Table B.10](#).

Offset	Key	Value	Description
0x00	Receiver Device	0x00	Host
0x01	Sender Device	0x6c	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x0026	(0x5+0x26 = 0x2b)
0x05	Receiver Object	0x00	
0x06	Sender Object	0x4e	
0x07	Packet ID	0x00	
0x08	Command	0x88	Socket Receive Request
0x09	Descriptor	0x09	
0x0a	Padding	0x00	
0x0b	Data Length	0x001c	
0x0d	Padding	0x0000	
0x1f	Data	...	

Table B.9.: GPS Session: Socket Receive Request

Offset	Key	Value	Description
0x00	Receiver Device	0x6c	Host
0x01	Sender Device	0x00	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x000e	(0x5+0x6 = 0xb)
0x05	Receiver Object	0x62	
0x06	Sender Object	0x1b	
0x07	Packet ID	0x00	
0x08	Command	0x7d	GPS Status Indication
0x09	Power	0x01	Enabled
0x0a	Status	0x01	Searching

Table B.10.: GPS Session: GPS Status Indication: No Lock

Next the Modem will start to send out GPS data packets with positioning information. Since it has not yet acquired a GPS satellite lock, it falls back to position information from the cellular network (e.g. center of Germany with accuracy being rough enough to cover the whole nation). Then, once the A-GPS results have been received, it will be a bit more accurate (the accuracy radius is < 5km in Germany). When enough GPS satellites have been found, another indication will be sent by the modem as visible in [Table B.11](#).

Offset	Key	Value	Description
0x00	Receiver Device	0x6c	Host
0x01	Sender Device	0x00	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x000e	(0x5+0x6 = 0xb)
0x05	Receiver Object	0x62	
0x06	Sender Object	0x1b	
0x07	Packet ID	0x00	
0x08	Command	0x91	GPS Status Indication
0x09	Power	0x01	Enabled
0x0a	Status	0x02	Lock acquired

Table B.11.: GPS Session: GPS Status Indication: Lock Acquired

Now the modem will sent out position information about once every second. As visible in the simplified [Table B.12](#), the GPS resource does not only provide the position, but also time information, a list of all visible satellites, the GSM or UMTS cell the user is currently connected to, as well as movement information derived from the current and the last position.

Offset	Key	Value	Description
0x00	Receiver Device	0x6c	Host
0x01	Sender Device	0x00	Modem
0x02	Resource	0x54	GPS
0x03	Length	0x00f2	(0x5+0xf2 = 0xf7)
0x05	Receiver Object	0x62	
0x06	Sender Object	0x1b	
0x07	Packet ID	0x00	
0x08	Command	0x92	GPS Data
0x09-0x0e	Padding	...	
0x0f	Number of sub-packets	0x05	(5 sub-packets)
0x10-0x12	Padding	...	
0x13-0x22	————— GPS time & date sub-packet —————		
0x13-0x22	...		
0x23-0x3e	————— GPS position sub-packet —————		
0x23	sub-packet marker	0x09	
0x24	sub-packet type ID	0x02	GPS Position
0x25-26	sub-packet length	0x001c	
0x27-0x2a	latitude	0x25cb0fbd	53.14670
0x2b-0x2e	longitude	0x05d1517e	8.18123
0x2f-0x3e	...		
0x3f-0x52	————— GPS movement sub-packet —————		
0x3f-0x52	...		
0x53-0xea	————— GPS satellite info sub-packet —————		
0x53-0xea	...		
0xeb-0xf6	————— GPS cell information sub-packet —————		
0xeb-0xf6	...		

Table B.12.: GPS Session: Hybrid Location Data

The protocol can be analyzed using the attached Wireshark plugin. The project has been started a couple of years ago and acquired full the GPS dissection capabilities during this thesis. [Figure B.2](#) depicts Wireshark with the loaded plugin dissecting the GPS session described above. For recording any packets send between the Modem and the CPU, *tcpdump* can be used in Maemo (available from the *fremantle-extra* repository).

The image displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, Internals, and Help. Below the menu is a toolbar with various icons for file operations and analysis. A filter bar is present with a text input field and buttons for Expression..., Clear, Apply, and Save.

The main display area is divided into three panes:

- Packet List:** A table showing 17 captured packets. The selected packet (No. 7) is highlighted in blue. The columns are No., Time, Protocol, Source, Destination, Resource, and Information.
- Packet Bytes:** A hex and ASCII view of the selected packet's raw data.
- Packet Details:** A hierarchical tree view of the selected packet's structure. It shows the Linux cooked capture, Intelligent Service Interface, Payload, and three subpackets: LS\_SB\_NPE\_POSITION, LS\_SB\_CELL\_INFO\_WCDMA, and another LS\_SB\_NPE\_POSITION.

The status bar at the bottom indicates: Frame (frame), 83 bytes | (Packets: 88 - Displayed: 88 (100.0%) - Load time: 0:00.000)

No.	Time	Protocol	Source	Destination	Resource	Information
1	17:29:56.496581000	ISI	Host	Modem	ComMgr,GPS	Subscribed Resources Indication
2	17:29:56.528930000	ISI	Host	Modem	GPS	Hybrid Tracking Request
3	17:29:56.530548000	ISI	Modem	Host	GPS	Hybrid Tracking Response
4	17:29:56.534759000	ISI	Modem	Host	GPS	Socket Open Request
5	17:29:57.951721000	ISI	Modem	Host	GPS	GPS Status Indication: GPS_STATE_SEARCH
6	17:29:57.951843000	ISI	Modem	Host	GPS	GPS Status Indication: GPS_STATE_SEARCH
7	17:29:57.951934000	ISI	Modem	Host	GPS	Hybrid Tracking Notification
8	17:29:58.927825000	ISI	Modem	Host	GPS	Hybrid Tracking Notification
9	17:29:59.321807000	ISI	Host	Modem	GPS	Socket Open Response
10	17:29:59.325042000	ULP	Modem	Host	GPS	msSPLSTART
11	17:29:59.326507000	ISI	Host	Modem	GPS	Socket Send Response
12	17:29:59.450347000	ULP	Host	Modem	GPS	msSPLRESPONSE
13	17:29:59.450927000	ISI	Modem	Host	GPS	Socket Receive Response
14	17:29:59.452056000	ULP	Modem	Host	GPS	msSPLPOSINIT
15	17:29:59.453979000	ISI	Host	Modem	GPS	Socket Send Response
16	17:29:59.023797000	ISI	Modem	Host	GPS	Hybrid Tracking Notification
17	17:29:59.024430000	ISI	Host	Modem	GPS	Socket Open Request

Figure B.2.: GPS Session in Wireshark

## C. Accelerometer Patches

This chapter provides the patches, which were required to complete the lis3lv02d driver's [DT](#) support regarding the Nokia N900's accelerometer. Additionally the last patch adds the lis3lv02d to the Nokia N900's device tree file. All four patches have been added to the mainline Linux kernel in 4.1-rc1.

### lis3lv02d: DT: use s32 to support negative values

The first patch required for the Nokia N900's accelerometer adds support for describing inverted axis to the existing DT axis description parser by reading it as signed integer instead of as unsigned integer. Support for reading signed integers from [DT](#) has been added to the kernel right before to this thesis. At the same time the patch adds support for negative thresholds, which are also used by the Nokia N900.

In the mainline Linux kernel this patch can be found under its shortened commit ID `cdcd6f824ecb`.

Listing C.1: [PATCH] lis3lv02d: DT: use s32 to support negative values

```
1 diff --git a/drivers/misc/lis3lv02d/lis3lv02d.c b/drivers/misc/lis3lv02d/
  lis3lv02d.c
2 index 3ef4627..d2b0968 100644
3 --- a/drivers/misc/lis3lv02d/lis3lv02d.c
4 +++ b/drivers/misc/lis3lv02d/lis3lv02d.c
5 @@ -950,6 +950,7 @@ int lis3lv02d_init_dt(struct lis3lv02d *lis3)
6     struct lis3lv02d_platform_data *pdata;
7     struct device_node *np = lis3->of_node;
8     u32 val;
9 +   s32 sval;
10
11     if (!lis3->of_node)
12         return 0;
13 @@ -1054,29 +1055,29 @@ int lis3lv02d_init_dt(struct lis3lv02d *lis3)
14     if (of_get_property(np, "st,hipass2-disable", NULL))
15         pdata->hipass_ctrl |= LIS3_HIPASS2_DISABLE;
16
17 -   if (of_get_property(np, "st,axis-x", &val))
18 -       pdata->axis_x = val;
19 -   if (of_get_property(np, "st,axis-y", &val))
20 -       pdata->axis_y = val;
21 -   if (of_get_property(np, "st,axis-z", &val))
22 -       pdata->axis_z = val;
23 +   if (of_property_read_s32(np, "st,axis-x", &sval) == 0)
24 +       pdata->axis_x = sval;
25 +   if (of_property_read_s32(np, "st,axis-y", &sval) == 0)
26 +       pdata->axis_y = sval;
27 +   if (of_property_read_s32(np, "st,axis-z", &sval) == 0)
28 +       pdata->axis_z = sval;
29
```

```

30     if (of_get_property(np, "st,default-rate", NULL))
31         pdata->default_rate = val;
32
33 -   if (of_get_property(np, "st,min-limit-x", &val))
34 -       pdata->st_min_limits[0] = val;
35 -   if (of_get_property(np, "st,min-limit-y", &val))
36 -       pdata->st_min_limits[1] = val;
37 -   if (of_get_property(np, "st,min-limit-z", &val))
38 -       pdata->st_min_limits[2] = val;
39 -
40 -   if (of_get_property(np, "st,max-limit-x", &val))
41 -       pdata->st_max_limits[0] = val;
42 -   if (of_get_property(np, "st,max-limit-y", &val))
43 -       pdata->st_max_limits[1] = val;
44 -   if (of_get_property(np, "st,max-limit-z", &val))
45 -       pdata->st_max_limits[2] = val;
46 +   if (of_property_read_s32(np, "st,min-limit-x", &sval) == 0)
47 +       pdata->st_min_limits[0] = sval;
48 +   if (of_property_read_s32(np, "st,min-limit-y", &sval) == 0)
49 +       pdata->st_min_limits[1] = sval;
50 +   if (of_property_read_s32(np, "st,min-limit-z", &sval) == 0)
51 +       pdata->st_min_limits[2] = sval;
52 +
53 +   if (of_property_read_s32(np, "st,max-limit-x", &sval) == 0)
54 +       pdata->st_max_limits[0] = sval;
55 +   if (of_property_read_s32(np, "st,max-limit-y", &sval) == 0)
56 +       pdata->st_max_limits[1] = sval;
57 +   if (of_property_read_s32(np, "st,max-limit-z", &sval) == 0)
58 +       pdata->st_max_limits[2] = sval;
59
60
61     lis3->pdata = pdata;

```

## lis3lv02d: DT: add wakeup unit 2 and wakeup threshold

The second patch required for the Nokia N900's accelerometer adds support for the second wakeup unit of the lis3lv02d. It is used by the N900 for receiving events on the Z-axis, while the first engine is configured to fire when the X or Y axis reach a specific threshold.

In the mainline Linux kernel this patch can be found under its shortened commit ID c5131a373613.

### Listing C.2: [PATCH] lis3lv02d: DT: add wakeup unit 2 and wakeup threshold

```

1 diff --git a/drivers/misc/lis3lv02d/lis3lv02d.c b/drivers/misc/lis3lv02d/
  lis3lv02d.c
2 index d2b0968..4739689 100644
3 --- a/drivers/misc/lis3lv02d/lis3lv02d.c
4 +++ b/drivers/misc/lis3lv02d/lis3lv02d.c
5 @@ -1032,6 +1032,23 @@ int lis3lv02d_init_dt(struct lis3lv02d *lis3)
6     pdata->wakeup_flags |= LIS3_WAKEUP_Z_LO;
7     if (of_get_property(np, "st,wakeup-z-hi", NULL))
8         pdata->wakeup_flags |= LIS3_WAKEUP_Z_HI;
9 +   if (of_get_property(np, "st,wakeup-threshold", &val))

```

## C. Accelerometer Patches

```
10 +     pdata->wakeup_thresh = val;
11 +
12 +     if (of_get_property(np, "st,wakeup2-x-lo", NULL))
13 +         pdata->wakeup_flags2 |= LIS3_WAKEUP_X_LO;
14 +     if (of_get_property(np, "st,wakeup2-x-hi", NULL))
15 +         pdata->wakeup_flags2 |= LIS3_WAKEUP_X_HI;
16 +     if (of_get_property(np, "st,wakeup2-y-lo", NULL))
17 +         pdata->wakeup_flags2 |= LIS3_WAKEUP_Y_LO;
18 +     if (of_get_property(np, "st,wakeup2-y-hi", NULL))
19 +         pdata->wakeup_flags2 |= LIS3_WAKEUP_Y_HI;
20 +     if (of_get_property(np, "st,wakeup2-z-lo", NULL))
21 +         pdata->wakeup_flags2 |= LIS3_WAKEUP_Z_LO;
22 +     if (of_get_property(np, "st,wakeup2-z-hi", NULL))
23 +         pdata->wakeup_flags2 |= LIS3_WAKEUP_Z_HI;
24 +     if (of_get_property(np, "st,wakeup2-threshold", &val))
25 +         pdata->wakeup_thresh2 = val;
26
27     if (!of_property_read_u32(np, "st,highpass-cutoff-hz", &val)) {
28         switch (val) {
```

### Documentation: DT: lis302: update wakeup binding

The third patch documents the API changes regarding the lis3lv02d DT binding in its binding document. It can be found in the mainline kernel via its shortened commit ID 21e8681862a5.

Listing C.3: [PATCH] Documentation: DT: lis302: update wakeup binding

```
1 diff --git a/Documentation/devicetree/bindings/misc/lis302.txt b/
   Documentation/devicetree/bindings/misc/lis302.txt
2 index 6def86f..2a19bff 100644
3 --- a/Documentation/devicetree/bindings/misc/lis302.txt
4 +++ b/Documentation/devicetree/bindings/misc/lis302.txt
5 @@ -46,11 +46,18 @@ Optional properties for all bus drivers:
6     interrupt 2
7     - st,wakeup-{x,y,z}-{lo,hi}:    set wakeup condition on x/y/z axis for
8         upper/lower limit
9     + - st,wakeup-threshold:        set wakeup threshold
10    + - st,wakeup2-{x,y,z}-{lo,hi}:  set wakeup condition on x/y/z axis for
11    +     upper/lower limit for second wakeup
12    +     engine.
13    + - st,wakeup2-threshold:        set wakeup threshold for second wakeup
14    +     engine.
15    - st,highpass-cutoff-hz=:    1, 2, 4 or 8 for 1Hz, 2Hz, 4Hz or 8Hz of
16        highpass cut-off frequency
17    - st,hipass{1,2}-disable:    disable highpass 1/2.
18    - st,default-rate=:          set the default rate
19    - - st,axis-{x,y,z}=:         set the axis to map to the three coordinates
20    + - st,axis-{x,y,z}=:         set the axis to map to the three coordinates.
21    +     Negative values can be used for inverted axis.
22    - st,{min,max}-limit-{x,y,z}  set the min/max limits for x/y/z axis
23        (used by self-test)
```



## DTS: ARM: OMAP3-N900: Add lis3lv02d support

Last but not least the fourth patch adds the lis3lv02d node to the Nokia N900's [DT](#) description file. It makes use of the previously introduced support for the second wakeup engine and inverts the Y and the Z axis as well as using negative threshold for the X axis.

It can be found in the mainline kernel via its shortened commit ID 1ac4e6fee41d.

Listing C.4: [PATCH] DTS: ARM: OMAP3-N900: Add lis3lv02d support

```
1 diff --git a/arch/arm/boot/dts/omap3-n900.dts b/arch/arm/boot/dts/omap3-
  n900.dts
2 index db80f9d..2cab149 100644
3 --- a/arch/arm/boot/dts/omap3-n900.dts
4 +++ b/arch/arm/boot/dts/omap3-n900.dts
5 @@ -609,6 +609,58 @@
6     pinctrl-0 = <&i2c3_pins>;
7
8     clock-frequency = <400000>;
9 +
10 + lis302d1: lis3lv02d@1d {
11 +     compatible = "st,lis3lv02d";
12 +     reg = <0x1d>;
13 +
14 +     Vdd-supply = <&vaux1>;
15 +     Vdd_IO-supply = <&vio>;
16 +
17 +     interrupt-parent = <&gpio6>;
18 +     interrupts = <21 20>; /* 181 and 180 */
19 +
20 +     /* click flags */
21 +     st,click-single-x;
22 +     st,click-single-y;
23 +     st,click-single-z;
24 +
25 +     /* Limits are 0.5g * value */
26 +     st,click-threshold-x = <8>;
27 +     st,click-threshold-y = <8>;
28 +     st,click-threshold-z = <10>;
29 +
30 +     /* Click must be longer than time limit */
31 +     st,click-time-limit = <9>;
32 +
33 +     /* Kind of debounce filter */
34 +     st,click-latency = <50>;
35 +
36 +     /* Interrupt line 2 for click detection */
37 +     st,irq2-click;
38 +
39 +     st,wakeup-x-hi;
40 +     st,wakeup-y-hi;
41 +     st,wakeup-threshold = <(800/18)>; /* millig-value / 18 to get HW
  values */
42 +
43 +     st,wakeup2-z-hi;
```

### C. Accelerometer Patches

```
44 +     st,wakeup2-threshold = <(900/18)>; /* millig-value / 18 to get HW
      values */
45 +
46 +     st,hipass1-disable;
47 +     st,hipass2-disable;
48 +
49 +     st,axis-x = <1>; /* LIS3_DEV_X */
50 +     st,axis-y = <(-2)>; /* LIS3_INV_DEV_Y */
51 +     st,axis-z = <(-3)>; /* LIS3_INV_DEV_Z */
52 +
53 +     st,min-limit-x = <(-32)>;
54 +     st,min-limit-y = <3>;
55 +     st,min-limit-z = <3>;
56 +
57 +     st,max-limit-x = <(-3)>;
58 +     st,max-limit-y = <32>;
59 +     st,max-limit-z = <32>;
60 + };
61 };
62
63 &mmc1 {
```

## D. OpenSCAD code

The 3D rendering of the 2D vector information for the layer cutter has been generated by exporting the 2D data into multiple files encoded in AutoCAD's Drawing Exchange Format. Next the files were imported in OpenSCAD, extruded to their correct thickness, slightly moved apart from each other and finally colored differently using the code from [Listing D.1](#).

Listing D.1: OpenSCAD-Code for 3D-rendering of the Serial-Adapter

```
1 /* file, pos, heighth, color (r,g,b,a) */
2 layers = [
3   ["layer0.dxf", 0, 3, [0.5,0.5,0.5,1]],
4   ["layer1.dxf", 20, 3, [0.5,0.5,0.5,1]],
5   ["layer2.dxf", 40, 6, [0.5,0.5,0.5,1]],
6   ["layer3.dxf", 55, 1.5, [1,0,0,1]], /* PCB */
7   ["layer4.dxf", 68, 2, [0.5,0.5,0.5,1]],
8   ["layer5.dxf", 85, 3, [0.5,0.5,0.5,1]],
9   ["layer6.dxf", 105, 14, [0.5,0.5,0.5,1]],
10  ["layer7.dxf", 125, 3, [0.5,0.5,0.5,1]],
11  ["layer8.dxf", 140, 2, [0.5,0.5,0.5,1]],
12 ];
13
14 for (layer = layers)
15   color(layer[3])
16     translate ([0,0,layer[1]])
17       linear_extrude(height = layer[2], center = true, convexity = 10)
18         import(file = layer[0]);
```

## E. Bill of Materials

To build the UART adapter described in [chapter 3](#) the components from [Table E.1](#) have been used.

Amount	PCB-ID	Name	Supplier	Unit Price	Total Price
10	—	PCB	Elecrow	-	9.88 €
1	C1	NPO-G0805 100P	Reichelt	0.05 €	0.04 €
3	C2-C4	X7R-G0805 1,0/25X7R-G0805 1,0/25	Reichelt	0.05 €	0.15 €
1	C5	SMD Tantal-Eko 4.7 uF (T491B475K016AT)	Distrelec	0.39 €	0.39 €
1	D1	SMD-LED 0805 GN	Reichelt	0.08 €	0.08 €
1	D2	SMD-LED 0805 RT	Reichelt	0.08 €	0.08 €
1	D3	SMD-LED 0805 BL	Reichelt	0.20 €	0.20 €
1	J1	USB BWM SMDUSB BWM SMD	Reichelt	0.27 €	0.27 €
1	K1	PicoBlade 90° SMD Pin Header (53261-0371)	Distrelec	0.47 €	0.47 €
1	U1	CY7C65215-32LTXI	Cypress	0.33 €	0.33 €
1	U2	INA219BIDCNT	RS Electronics	2.29 €	2.29 €
1	U3	TPS76918	Distrelec	0.68 €	0.68 €
6	P1-P6	GKS181	Ebay	≈ 1.00 €	6.00 €
1	Q1	FDN338P	Distrelec	0.40 €	0.40 €
2	Q2, Q3	BSS 138 SMD	Reichelt	0.04 €	0.08 €
3	R1, R2, R13	SMD-0805 220	Reichelt	0.11 €	0.33 €
3	R3, R4, R9	SMD-0805 2,20K	Reichelt	0.11 €	0.33 €
1	R5	SMD Resistor 0.02 (LR1206-21R020FA)	Distrelec	0.33 €	0.33 €
2	R6, R7	SMD-0805 10,0	Reichelt	0.11 €	0.22 €
4	R8, R10-R12	SMD-0805 10,0K	Reichelt	0.11 €	0.44 €
					23.71 €

Table E.1.: Bill of Materials for UART Adapter PCB

## F. Simple Ofono Dialer

Listing F.1 contains the simple dialing application, that has been written to test the functionality of the lower stack. A description of its functionality and Screenshots can be found in [chapter 6](#).

Listing F.1: Simple Ofono dialing and short message application

```
1 #!/usr/bin/python3
2 from dbus.mainloop.glib import DBusGMainLoop
3 from gi.repository import Gtk, GObject
4 import dbus, dbus.service
5 import sys, time
6
7 class PINDialog(Gtk.Dialog):
8     def __init__(self, parent = None):
9         Gtk.Dialog.__init__(self, "SIM PIN", parent, 0,
10                             (Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
11                              Gtk.STOCK_OK, Gtk.ResponseType.OK))
12         self.set_modal(True)
13
14         self.grid = Gtk.Grid()
15         self.get_content_area().add(self.grid)
16
17         self.label = Gtk.Label()
18         self.label.set_justify(Gtk.Justification.CENTER)
19         self.label.set_markup("<span font_desc=\"32.0\">Pin Required</span>")
20         self.grid.attach(self.label, 0, 0, 1, 1)
21
22         self.pin_entry = Gtk.Entry()
23         self.pin_entry.connect("activate", self.__emit_ok)
24         self.pin_entry.set_expand(True)
25         self.pin_entry.set_visibility(False)
26         self.grid.attach(self.pin_entry, 0, 1, 1, 1)
27
28         self.show_all()
29
30     def __emit_ok(self, widget):
31         self.emit("response", Gtk.ResponseType.OK)
32
33     def get_pin(self):
34         return self.pin_entry.get_text()
35
36 class SMSReceiveDialog(Gtk.Dialog):
37     def __init__(self, parent, number, time, message):
38         Gtk.Dialog.__init__(self, "SMS", parent, 0,
39                             (Gtk.STOCK_OK, Gtk.ResponseType.OK))
40         self.set_modal(True)
41         box = self.get_content_area()
42
43         self.num_label = Gtk.Label()
44         self.num_label.set_markup("<b>Phone Number:</b> " + number)
45         box.add(self.num_label)
46
47         self.time_label = Gtk.Label()
48         self.time_label.set_markup("<b>Time:</b> " + time)
49         box.add(self.time_label)
50
51         self.sw = Gtk.ScrolledWindow()
52         self.sw.set_policy(Gtk.PolicyType.AUTOMATIC, Gtk.PolicyType.AUTOMATIC)
53
54         self.textbox = Gtk.TextView()
55         self.textbox.set_editable(False)
```

## F. Simple Ofono Dialer

```
56         self.textbox.set_hexpand(True)
57         self.textbox.set_vexpand(True)
58
59         self.sw.add(self.textbox)
60
61         buf = self.textbox.get_buffer()
62         buf.set_text(message)
63
64         box.add(self.sw)
65         self.show_all()
66
67     class SMSSendDialog(Gtk.Dialog):
68         def __init__(self, parent):
69             Gtk.Dialog.__init__(self, "SMS", parent, 0, ())
70             self.set_modal(True)
71
72             cancel_button = self.add_button("Cancel", Gtk.ResponseType.CANCEL)
73             image = Gtk.Image()
74             image.set_from_icon_name("gtk-cancel", Gtk.IconSize.BUTTON)
75             cancel_button.set_always_show_image(True)
76             cancel_button.set_image(image)
77
78             send_button = self.add_button("Send", Gtk.ResponseType.OK)
79             image = Gtk.Image()
80             image.set_from_icon_name("mail-send", Gtk.IconSize.BUTTON)
81             send_button.set_always_show_image(True)
82             send_button.set_image(image)
83
84             self.sw = Gtk.ScrolledWindow()
85             self.sw.set_policy(Gtk.PolicyType.AUTOMATIC, Gtk.PolicyType.AUTOMATIC)
86
87             self.textbox = Gtk.TextView()
88             self.textbox.set_editable(True)
89             self.textbox.set_hexpand(True)
90             self.textbox.set_vexpand(True)
91
92             self.sw.add(self.textbox)
93
94             box = self.get_content_area()
95             box.add(self.sw)
96             self.show_all()
97
98         def get_message(self):
99             buf = self.textbox.get_buffer()
100             start = buf.get_start_iter()
101             stop = buf.get_end_iter()
102             return buf.get_text(start, stop, True)
103
104     class CallDialog(Gtk.Dialog):
105         def __init__(self, parent, number):
106             Gtk.Dialog.__init__(self, "Call", parent, 0, ())
107             self.set_modal(True)
108
109             hangup_button = self.add_button("Hangup", Gtk.ResponseType.CANCEL)
110             image = Gtk.Image()
111             image.set_from_icon_name("call-stop", Gtk.IconSize.BUTTON)
112             hangup_button.set_always_show_image(True)
113             hangup_button.set_image(image)
114
115             self.number = number
116             self.label = Gtk.Label()
117             self.label.set_justify(Gtk.Justification.CENTER)
118             msg = "<span font_desc=\"32.0\">Active Voice Call\n<b>"+number+"</b></span>"
119             self.label.set_markup(msg)
120             box = self.get_content_area()
121             box.add(self.label)
122
123             self.show_all()
124
125     class IncomingCallDialog(Gtk.Dialog):
126         def __init__(self, parent, number):
```

```

127     Gtk.Dialog.__init__(self, "Call", parent, 0, ())
128     self.set_modal(True)
129
130     hangup_button = self.add_button("Hangup", Gtk.ResponseType.CANCEL)
131     image = Gtk.Image()
132     image.set_from_icon_name("call-stop", Gtk.IconSize.BUTTON)
133     hangup_button.set_always_show_image(True)
134     hangup_button.set_image(image)
135
136     accept_button = self.add_button("Accept", Gtk.ResponseType.ACCEPT)
137     image = Gtk.Image()
138     image.set_from_icon_name("call-start", Gtk.IconSize.BUTTON)
139     accept_button.set_always_show_image(True)
140     accept_button.set_image(image)
141
142     self.number = number
143     self.label = Gtk.Label()
144     self.label.set_justify(Gtk.Justification.CENTER)
145     msg = "<span font_desc=\"32.0\">Incoming Voice Call\n<b>"+self.number+"</b></span>"
146     self.label.set_markup(msg)
147     box = self.get_content_area()
148     box.add(self.label)
149
150     self.show_all()
151
152 class DialerWindow(Gtk.Window):
153     __gsignals__ = {
154         'send-sms': (GObject.SIGNAL_RUN_FIRST, None, (str, str, )),
155         'start-voice-call': (GObject.SIGNAL_RUN_FIRST, None, (str, ))
156     }
157
158     def __init__(self):
159         Gtk.Window.__init__(self, title="Dialer")
160
161         self.grid = Gtk.Grid()
162         self.grid.set_hexpand(True)
163         self.add(self.grid)
164
165         buttons = [
166             ["1", 0, 0], ["2", 1, 0], ["3", 2, 0],
167             ["4", 0, 1], ["5", 1, 1], ["6", 2, 1],
168             ["7", 0, 2], ["8", 1, 2], ["9", 2, 2],
169             ["*", 0, 3], ["0", 1, 3], ["#", 2, 3],
170         ]
171
172         for bdesc in buttons:
173             button = Gtk.Button(label=bdesc[0])
174             button.set_size_request(100,100)
175             button.connect("clicked", self.on_number_button_clicked)
176             self.grid.attach(button, bdesc[1], bdesc[2], 1, 1)
177
178         self.number_entry = Gtk.Entry()
179         self.number_entry.set_hexpand(True)
180         self.grid.attach(self.number_entry, 4, 0, 1, 1)
181
182         image = Gtk.Image()
183         image.set_from_stock(Gtk.STOCK_GO_BACK, Gtk.IconSize.LARGE_TOOLBAR)
184         self.buttonBack = Gtk.Button()
185         self.buttonBack.set_image(image)
186         self.buttonBack.set_size_request(100,100)
187         self.buttonBack.connect("clicked", self.on_back_button_clicked)
188         self.grid.attach(self.buttonBack, 5, 0, 1, 1)
189
190         image = Gtk.Image()
191         image.set_from_icon_name("call-start", Gtk.IconSize.LARGE_TOOLBAR)
192         self.buttonCall = Gtk.Button()
193         self.buttonCall.set_image(image)
194         self.buttonCall.set_size_request(100,100)
195         self.buttonCall.connect("clicked", self.on_call_button_clicked)
196         self.grid.attach(self.buttonCall, 4, 1, 2, 1)
197

```

## F. Simple Ofono Dialer

```
198     image = Gtk.Image()
199     image.set_from_icon_name("mail-message-new", Gtk.IconSize.LARGE_TOOLBAR)
200     self.buttonSMS = Gtk.Button()
201     self.buttonSMS.set_image(image)
202     self.buttonSMS.set_size_request(100,100)
203     self.buttonSMS.connect("clicked", self.on_sms_button_clicked)
204     self.grid.attach(self.buttonSMS, 4, 2, 2, 1)
205
206     image = Gtk.Image()
207     image.set_from_stock(Gtk.STOCK_QUIT, Gtk.IconSize.LARGE_TOOLBAR)
208     self.buttonQuit = Gtk.Button()
209     self.buttonQuit.set_image(image)
210     self.buttonQuit.set_size_request(100,100)
211     self.buttonQuit.connect("clicked", self.on_quit_button_clicked)
212     self.grid.attach(self.buttonQuit, 4, 3, 2, 1)
213
214     self.show_all()
215
216     def on_number_button_clicked(self, widget):
217         new = self.number_entry.get_text() + widget.get_label()
218         self.number_entry.set_text(new)
219
220     def on_back_button_clicked(self, widget):
221         new = self.number_entry.get_text()[:-1]
222         self.number_entry.set_text(new)
223
224     def on_call_button_clicked(self, widget):
225         number = self.number_entry.get_text()
226         if number == "":
227             return
228         self.emit("start-voice-call", number)
229
230     def on_sms_button_clicked(self, widget):
231         number = self.number_entry.get_text()
232         if number == "":
233             return
234         dialog = SMSSendDialog(self)
235         response = dialog.run()
236         if response == Gtk.ResponseType.OK:
237             message = dialog.get_message()
238             dialog.destroy()
239             self.emit("send-sms", number, message)
240
241     def on_quit_button_clicked(self, widget):
242         Gtk.main_quit()
243
244     class OfonoController:
245         def __init_dbus_connection__(self):
246             try:
247                 DBusGMainLoop(set_as_default=True)
248                 self.bus = dbus.SystemBus()
249                 self.ofono = self.bus.get_object('org.ofono', '/n900_0')
250                 self.modem = dbus.Interface(self.ofono, 'org.ofono.Modem')
251                 self.sim = dbus.Interface(self.ofono, 'org.ofono.SimManager')
252                 self.sms = dbus.Interface(self.ofono, 'org.ofono.MessageManager')
253                 self.call = dbus.Interface(self.ofono, 'org.ofono.VoiceCallManager')
254             except dbus.exceptions.DBusException as e:
255                 dialog = Gtk.MessageDialog(None, 0, Gtk.MessageType.ERROR,
256                     Gtk.ButtonsType.CLOSE, "DBus Exception")
257                 dialog.format_secondary_text(str(e))
258                 dialog.run()
259                 dialog.destroy()
260                 sys.exit(1)
261
262         def __init_ofono__(self):
263             self.modem.SetProperty("Powered", True)
264
265             if self.sim.GetProperties()["PinRequired"] != "none":
266                 dialog = PINDialog()
267                 response = dialog.run()
268                 pin = dialog.get_pin()
```



```

269         dialog.destroy()
270         if response != Gtk.ResponseType.OK:
271             sys.exit(1)
272         print("Pin: \"%s\"" % pin)
273         self.sim.EnterPin("pin", pin)
274
275     time.sleep(3)
276
277     if self.sim.GetProperties()["PinRequired"] != "none":
278         dialog = Gtk.MessageDialog(None, 0, Gtk.MessageType.ERROR,
279             Gtk.ButtonsType.CLOSE, "Incorect PIN")
280         dialog.format_secondary_text("Provided PIN was not correct!")
281         dialog.run()
282         dialog.destroy()
283         sys.exit(1)
284
285     self.modem.SetProperty("Online", True)
286
287     self.call.connect_to_signal("CallAdded", self.on_ofono_call_added)
288     self.call.connect_to_signal("CallRemoved", self.on_ofono_call_removed)
289     self.sms.connect_to_signal("IncomingMessage", self.on_ofono_message)
290
291     def __init__(self):
292         self.__init_dbus_connection__()
293         self.__init_ofono__()
294
295         self.win = DialerWindow()
296         self.win.connect("delete-event", Gtk.main_quit)
297         self.win.connect("send-sms", self.on_send_sms)
298         self.win.connect("start-voice-call", self.on_start_voice_call)
299
300     def accept_call(self, path):
301         callobj = self.bus.get_object('org.ofono', path)
302         callinterface = dbus.Interface(callobj, 'org.ofono.VoiceCall')
303         callinterface.Answer()
304
305     def hangup_call(self, path):
306         try:
307             callobj = self.bus.get_object('org.ofono', path)
308             callinterface = dbus.Interface(callobj, 'org.ofono.VoiceCall')
309             callinterface.Hangup()
310         except dbus.exceptions.DBusException as e:
311             # path may no longer exist, if the remote side ended the call
312             pass
313
314     def on_send_sms(self, window, number, message):
315         print("Send SMS (%s): %s" % (number, message))
316         self.sms.SendMessage(number, message)
317
318     def on_start_voice_call(self, window, number):
319         print("Voice Call:", number)
320         self.call.Dial(number, False)
321
322     def on_ofono_call_removed(self, self, path):
323         print("End Voice Call!")
324         self.calldialog.response(Gtk.ResponseType.CANCEL)
325
326     def on_ofono_call_added(self, self, path, properties):
327         if properties["State"] == "incoming":
328             print("Incoming Call!")
329         else:
330             print("Outgoing Call!")
331
332         number = properties["LineIdentification"]
333
334         # Dialog shown for accepting/rejecting incoming calls
335         if properties["State"] == "incoming":
336             self.calldialog = IncomingCallDialog(self.win, number)
337             response = self.calldialog.run()
338             self.calldialog.destroy()
339

```

## F. Simple Ofono Dialer

```
340         if response != Gtk.ResponseType.ACCEPT:
341             self.hangup_call(path)
342             print("Incoming Call Rejected!")
343             return
344         else:
345             self.accept_call(path)
346             print("Incoming Call Accepted!")
347
348         # Dialog shown during active voice calls
349         self.calldialog = CallDialog(self.win, number)
350         response = self.calldialog.run()
351         self.calldialog.destroy()
352
353         self.hangup_call(path)
354
355         print("Call has been hung up!")
356
357     def on_ofono_message(self, message, info):
358         number = info["Sender"]
359         time = info["SentTime"]
360         print("Incoming Message (%s): %s" % (sender, message))
361         dialog = SMSReceiveDialog(self.win, sender, time, message)
362         dialog.run()
363         dialog.destroy()
364
365     if __name__ == "__main__":
366         settings = Gtk.Settings.get_default()
367         settings.set_long_property("gtk-entry-password-hint-timeout", 500, "ofono-dialer")
368
369         main = OfonoController()
370         Gtk.main()
```

## Bibliography

- [1] Ricardo Salveti de Araujo. *Ubuntu Touch Internals*. Apr. 2014. URL: [https://events.linuxfoundation.org/sites/events/files/slides/Ubuntu%20Touch%20Internals\\_1.pdf](https://events.linuxfoundation.org/sites/events/files/slides/Ubuntu%20Touch%20Internals_1.pdf).
- [2] Hans-Joachim Baader. *Imagination: Freier Grafiktreiber für PowerVR*. June 2015. URL: <http://www.pro-linux.de/news/1/22439/imagination-freier-grafiktreiber-fuer-powervr.html>.
- [3] Brian Benchoff. *DIY SMD stencils made with a craft cutter*. Dec. 2012. URL: <http://hackaday.com/2012/12/27/diy-smd-stencils-made-with-a-craft-cutter/>.
- [4] Tim Bird. “Overcoming Obstacles to Mainlining”. In: ELCE. 2014. URL: [http://www.elinux.org/images/8/8f/Overcoming\\_Obstacles\\_to\\_Mainlining-ELCE-2014-with-notes.pdf](http://www.elinux.org/images/8/8f/Overcoming_Obstacles_to_Mainlining-ELCE-2014-with-notes.pdf).
- [5] Neil Brown. *tty slave device support - version 3*. Mar. 2015. URL: <http://lkml.kernel.org/r/20150318055437.21025.13990.stgit@notabene.brown>.
- [6] Alison Chaiken. “Best practices for long-term support and security of the device-tree”. In: ELCE. 2013. URL: [http://events.linuxfoundation.org/sites/events/files/slides/DT\\_ELCE\\_2013.pdf](http://events.linuxfoundation.org/sites/events/files/slides/DT_ELCE_2013.pdf).
- [7] Carlos Chinaea. *HSI framework and drivers*. Apr. 2010. URL: <https://lwn.net/Articles/384526/>.
- [8] Jonathan Corbet. *Obstacles to contribution in embedded Linux*. June 2015. URL: <https://lwn.net/Articles/647524/>.
- [9] *FreeSmartPhone.org Project*. URL: <http://www.freesmartphone.org/>.
- [10] Peter Gewalt. “Sicherheitsaspekte von Mobiltelefonen - Erkennung und Visualisierung von Angriffsvektoren”. MA thesis. Department Informatik: Carl von Ossietzky University of Oldenburg, Oct. 2014.
- [11] Quim Gil. “How Maemo approaches Open Source”. In: Maemo Summit. 2008. URL: [https://maemo.org/midcom-serveattachmentguid-4c0c9590887911dd804ae36efcafed98ed98/how\\_nokia\\_approaches\\_open\\_source\\_for\\_maemo.pdf](https://maemo.org/midcom-serveattachmentguid-4c0c9590887911dd804ae36efcafed98ed98/how_nokia_approaches_open_source_for_maemo.pdf).
- [12] Bluetooth Special Interest Group. *Specification of the Bluetooth System: Host Controller Interface [Transport Layer]*. Jan. 2006. URL: [https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc\\_id=41266](https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=41266).
- [13] Marcel Holtmann. *Bluetooth: hci-uart: Add protocol support for Nokia UART devices*. Apr. 2015. URL: <http://permalink.gmane.org/gmane.linux.bluez.kernel/61384>.
- [14] Marcel Holtmann. *Bluetooth: hci-uart: Provide generic H:4 receive framework*. Apr. 2015. URL: <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=79b8df9362e8bd1951e1fddbd65ca87af8df52c8>.
- [15] *How is Ubuntu Touch connected to Android?* URL: [https://wiki.ubuntu.com/Touch/FAQ#How\\_is\\_Ubuntu\\_Touch\\_connected\\_to\\_Android.3F](https://wiki.ubuntu.com/Touch/FAQ#How_is_Ubuntu_Touch_connected_to_Android.3F).

- [16] Arasan Chip Systems Inc. *HSI Controller IP Core*. May 2015. URL: <http://arasan.com/products/mipi/hsi-controller/>.
- [17] Russell King. *ARM: proc-v7: avoid errata 430973 workaround for non-Cortex A8 CPUs*. Apr. 2015. URL: <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=a6d746789825e4d7229523eebee233b03ad48c54>.
- [18] Ingmar Kristian Klein. “ARM SOCs as a desktop replacement?” MA thesis. June 2014. URL: [http://www.hs-augsburg.de/~ingmar\\_k/Masterarbeit\\_WS1314/Vorabversion/PDF/preliminary.pdf](http://www.hs-augsburg.de/~ingmar_k/Masterarbeit_WS1314/Vorabversion/PDF/preliminary.pdf).
- [19] Luke Kenneth Casson Leighton. *PowerVR SGX USSE Opcodes*. June 2015. URL: <http://powervr.gnu.org/ve/doku.php?id=opcodes>.
- [20] Tony Lindgren. *[GIT PULL 1/2] omap clean-up for v4.2*. May 2015. URL: <https://patchwork.ozlabs.org/patch/474676/>.
- [21] Robert Love. *Linux Kernel development*. 3rd ed. Addison-Wesley Professional, 2010. ISBN: 978-0672329463.
- [22] *Maemo Wiki: N900 Hardware Hacking*. URL: [https://wiki.maemo.org/N900\\_Hardware\\_Hacking#Debug\\_ports](https://wiki.maemo.org/N900_Hardware_Hacking#Debug_ports).
- [23] Catalin Marinas. *[ARM] 5487/1: ARM errata: Stale prediction on replaced interworking branch*. Apr. 2009. URL: <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=7ce236fcd6fd45b0441a2d49acb2ceb2de2e8a47>.
- [24] Nishanth Menon. *ARM: Introduce erratum workaround for 430973*. Mar. 2015. URL: <http://git.denx.de/?p=u-boot.git;a=commitdiff;h=5902f4ce0f2bd1411e40dc0ec3598a0fc19b2ae>.
- [25] Microchip. *Application Note 1941*. May 2015. URL: <http://ww1.microchip.com/downloads/cn/AppNotes/cn574185.pdf>.
- [26] *Neo900 Project*. URL: <http://neo900.org/>.
- [27] Kantar World Panel. “Kantar: Apple gewinnt Marktanteile, Android verliert”. In: (Jan. 2015). URL: <http://www.heise.de/mac-and-i/meldung/Kantar-Apple-gewinnt-Marktanteile-Android-verliert-2513813.html>.
- [28] Thomas Petazzoni. “Device Tree for Dummies”. In: ELCE. 2013. URL: <http://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf>.
- [29] *Pocket-Computer Nokia 770 Internet Tablet und die Maemo Development Platform*. July 2005. URL: <http://www.linux-magazin.de/Ausgaben/2005/07/Aus-heiterem-Himmel>.
- [30] Jürgen Quade and Eva-Katharina Kunst. *Linux-Treiber entwickeln: Eine systematische Einführung in die Gerätetreiber- und Kernelprogrammierung*. 2nd ed. dpunkt.verlag, 2006. ISBN: 978-3898646963.
- [31] Sebastian Reichel. *ARM: OMAP2+: Add support for thumb mode on DT booted N900*. Feb. 2014. URL: <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=deff82e688a278eb4b822fd616c05fc62907bb71>.
- [32] Martin Sauter. *Grundkurs Mobile Kommunikationssysteme: UMTS, HSDPA und LTE, GSM, GPRS und Wireless LAN*. 5th ed. Aug. 2013. ISBN: 978-3658014605.
- [33] Freescale Semiconductor. *AN3397: Implementing Positioning Algorithms Using Accelerometers*. Feb. 2007. URL: [https://www.freescale.com/files/sensors/doc/app\\_note/AN3397.pdf](https://www.freescale.com/files/sensors/doc/app_note/AN3397.pdf).

- [34] Freescale Semiconductor. *AN3461: Tilt Sensing Using a Three-Axis Accelerometer*. Mar. 2013. URL: [https://cache.freescale.com/files/sensors/doc/app\\_note/AN3461.pdf](https://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf).
- [35] *Stable Hybrid Release*. URL: <http://www.shr-project.org/>.
- [36] Linus Torvalds. *Linux 4.0-rc1 out.*. Feb. 2015. URL: <https://lkml.org/lkml/2015/2/22/203>.
- [37] Linus Torvalds. *Merge branch 'for-next' of git://gitorious.org/kernel-hsi/kernel-hsi*. Apr. 2012. URL: <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/drivers/Makefile?id=b1a808ff436343956a6ae63178ea1810c5e5a3a1>.
- [38] Kai Vehmanen. *N900 Modem Speech Support*. Mar. 2015. URL: <https://lkml.org/lkml/2015/3/5/606>.
- [39] Linus Walleij. *HSI subsystem status*. Jan. 2013. URL: <https://lkml.org/lkml/2013/1/6/150>.

## Glossary

**8-N-1 format** 8 data bits, no parity bit, 1 stop bit. [21](#), [35](#)

**ACL** Asynchronous Connection-Less. [34](#)

**AEWB** Auto Exposure White Balance. [38](#)

**AF** Auto Focus. [38](#), [39](#)

**ALSA** Advanced Linux Sound Architecture. [31](#), [45](#), [52](#)

**Android** Linux distribution from Google with its main focus on smartphones and tablets. [2–4](#)

**API** Application Programming Interface. [24](#), [27](#), [31](#), [33](#), [44](#), [47](#), [53](#)

**ARM Holdings** British-based semiconductor and software design company. Its largest business is designing processors (CPU) bearing the ARM name. [6](#), [32](#)

**CCP2** Compact Camera Port 2. [36](#), [38](#), [39](#)

**chroot** program, which tells the operating system to use a sub-directory as new root node for all sub-processes. Can be used to switch into another Linux distribution. [4](#)

**CNC** Computerized Numerical Control. [14](#), [17](#)

**CNC-machine** A machine tool, that is controlled from a computer (e.g. a laser-cutter or a mill) see. [14](#)

**CPU** Central Processing Unit. [6](#), [25](#), [26](#), [32](#)

**CSI 1** Camera Serial Interface. [36](#)

**CTS** Clear to Send. [35](#)

**Cypress** semiconductor manufacturer, known for its USB chips. [15](#)

**D-Bus** IPC and RPC implementation designed as part of the freedesktop.org project. [44](#), [47](#)

**DAI** Digital Audio Interface. [27](#)

**Debian** Community driven Linux distribution. [3](#), [25](#), [32](#)

**DHCP** Dynamic Host Configuration Protocol. [30](#)

- DMA** Direct Memory Access. [6](#), [7](#), [9](#), [11](#), [12](#), [30](#)
- DSI** Display Serial Interface. [11](#)
- DSP** Digital Signal Processor. [26](#), [44](#)
- DSS** Display Subsystem. [11](#), [12](#), [25](#), [26](#)
- DT** device tree. [6](#), [7](#), [9](#), [25](#), [27–29](#), [31](#), [33](#), [35–39](#), [53](#), [70](#), [72](#), [73](#)
- EFL** Enlightenment Foundation Libraries. [3](#)
- Fairchild Semiconductor** Fairchild Semiconductor International, Inc., semiconductor manufacturer, among others known for its transistors. [15](#)
- FM** Frequency modulation. [33](#), [35](#), [52](#)
- fork** In Software Development a fork means, that the source code of a project is copied and developed on independently. After the fork there are two projects, which are worked on independently. [2](#)
- framebuffer** memory area containing a bitmap, that should be send to a video display. [25](#)
- FSO** freesmartphone.org. [3](#), [4](#), [44](#), [45](#)
- FTDI** Future Technology Devices International, commonly known for its USB-serial chips. [15](#)
- GNU** GNU is not unix. [55](#)
- GPIO** General Purpose Input/Output. [7](#), [9](#), [11](#), [12](#), [25](#), [26](#), [29](#), [30](#), [35](#), [36](#), [52](#), [59](#)
- GPL** GNU Public License. [53](#)
- GPRS** General Packet Radio Service. [30](#)
- GPS** Global Positioning System. [44](#), [63](#), [66](#), [67](#)
- GPU** processor for calculation of 2D and 3D graphical data. [25](#), [26](#), [53](#)
- Graphviz** open source graph visualization software (<http://www.graphviz.org>). [9](#)
- GSM** Global System for Mobile Communications. [28](#), [44](#), [66](#)
- GUI** Graphical User Interface. [3](#), [43–45](#)
- H:4** Bluetooth transport protocol for serial devices. [33](#), [35](#)
- HCI** Host Controller Interface. [33](#)

**HSI** Highspeed Serial Interface. [27](#), [29](#), [30](#)

**I2C** Inter-Integrated Circuit, serial bus, which supports multiple slaves devices and optionally multiple masters. Usually used for connecting in-device sensors to the processor.. [9](#), [10](#), [15](#), [25](#), [26](#), [28](#), [33](#), [51](#), [52](#)

**IMSI** International Mobile Subscriber Identity. [66](#)

**ioctl** input/output control, device-specific system-call related to input/output operations. [31](#)

**IOMMU** [MMU](#) for I/O devices. [26](#)

**IP** Internet Protocol. [30](#)

**IP-Core** A reusable unit (of logic), that is intellectual property and may be licensed, e.g. an MP3-encoder. [7](#), [9](#), [11](#), [25](#), [29](#), [30](#), [33](#), [53](#)

**IPC** inter-process communication. [42](#)

**ISI** Intelligent Service Interface. [5](#), [30](#), [44](#), [52](#), [62](#), [63](#)

**ISP** Image Signal Processor. [9](#), [26](#), [27](#), [36](#), [38](#), [53](#)

**kernel** fundamental part of modern operating systems, that acts as low-level abstraction layer. [2](#)

**LCD** Liquid Crystal Display. [11](#), [12](#)

**Linus Torvalds** originator of the Linux kernel project. [23](#), [24](#), [29](#)

**Linux** open source Unix-like operating system kernel. [2](#), [55](#), *see* [kernel](#)

**Linux distribution** common name for operating systems based on the Linux kernel, e.g. Debian or Red Hat Enterprise Linux. [2–4](#), [23](#), [25](#), [32](#), [33](#), [47](#), [53](#), [55](#), *see* [Linux](#) & [kernel](#)

**LIS3LV02D** Accelerometer Chip from STMicroeletronics with I2C or SPI interface <http://mipi.org>. [28](#)

**LKML** Linux kernel mailing list. [35](#), [36](#)

**Maemo** Linux distribution developed by Nokia for their Internet tablet hardware series, which gained smartphone capabilities in later versions. In 2010 it has been merged with Intel's Moblin and is known as MeeGo since then. . [3](#), [13](#)

**McBSP** Multichannel Buffered Serial Port. [27](#), [33](#)

**McSAAB** SSI protocol for communication with Nokia N900's modem, also known as improved SSI protocol or simply SSI protocol. [26](#), [30](#), [31](#), [43](#)



- MFD** multi function device. [36](#)
- Microsoft Corporation** American multinational corporation known for its Operating System Windows, Office Suite and the game console XBox. [3](#)
- MIPI** MIPI Alliance <http://mipi.org>. [11](#), [27](#)
- MMC** MultiMediaCard. [25](#)
- MMU** Memory Management Unit. [38](#), [88](#)
- N900** Smartphone developed by Nokia, which is using Texas Instruments OMAP3 processor (ARM based) and comes with Maemo by default. . [3](#), [5](#), [13](#), [15](#)
- Nokia Oyj** Finish communications and information technology company. [3](#)
- NOLO** bootloader used by the Nokia N900. [21](#)
- OMAP** Open Multimedia Applications Platform, ARM based SoC developed for smart phones and tablets. [6](#), [7](#), [12](#), [13](#), [25](#), [27](#), [29](#), [32](#), *see* [SoC](#)
- OpenGL** Open Graphics Language. [42](#), [50](#)
- OpenMoko** small company, which created a open smartphone running a custom Linux distribution. [2](#), [3](#), [44](#)
- PCB** Printed Circuit Board. [5](#), [15](#), [17](#), [19](#), [20](#), [29](#), [51](#), [52](#)
- PCM** Pulse-code modulation. [31](#)
- phandle** Numerical reference to another node in the Device Tree. [37](#)
- PhoNet** Phone Network protocol (PhoNet) is a simple binary protocol abstracting modem interconnect with support for multiplexing and asynchronous notifications. [26](#), [27](#), [30](#), [31](#), [43](#), [44](#), [52](#), [62](#)
- PLL** Phase Locked Loop, system to generate an output signal related to an incoming signal, commonly used to generate or distribute clock signals in embedded devices. [36](#)
- Pogo-Pin** A pin, which contains a spring, so that it can be compressed. E.g. used for temporary connections between two PCBs. [vi](#), [13](#), [19](#), [20](#), [51](#)
- QFN32** PCB footprint with 32 pads (no pins) at the chips sides. [17](#), [51](#)
- RAM** Random Access Memory. [27](#), [29](#)
- RISC** Reduced Instruction Set Computing. [32](#)

**RNG** Random Number Generator. [26](#)

**RS232** Standard for serial transmission defining some additional signals. [15](#)

**RTC** clock tracking absolute time values. [26](#)

**RTS** Ready to Send. [35](#)

**SCO** Synchronous connection-oriented. [34](#)

**SDI** Serial Display Interface. [11](#)

**SHR** Stable Hybrid Release. [3](#), [4](#), [44](#)

**SIM** Subscriber Identity Module. [47](#)

**SMD** Surface Mounted Device. [15](#), [17](#)

**SMIA** SMIA (Standard Mobile Imaging Architecture) is an image sensor standard defined jointly by Nokia and ST. SMIA++, defined by Nokia, is an extension of that. These definitions are valid for both types of sensors.. [36](#), [38](#)

**SoC** System on a Chip, IC integrating all components of a computer or similar system into a single chip. [6](#), [8](#), [9](#), [13](#), [28](#), [29](#), [53](#)

**SparkFun** Manufacturer of development and breakout boards. [15](#)

**SPI** Serial Peripheral Interface, full-duplex serial bus intended for embedded devices. [7](#), [8](#), [11](#), [12](#), [15](#), [26](#)

**SSI** Synchronous Serial Interface. [26](#), [27](#), [29–31](#)

**SUPL** Secure User Plane Location Protocol. [66](#)

**SVG** Scalable Vector Graphics. [14](#)

**Telepathy** software framework for instant messaging, voice over ip, video calls. [44](#)

**Texas Instruments** semiconductor manufacturer, among others responsible for the OMAP SoCs. [6](#), [10](#), [11](#), [15](#), [27](#), [53](#)

**Thumb** compact 16-bit encoded subset of the ARM instruction set to reduce required memory space. [32](#)

**U-Boot** bootloader for embedded systems - <http://www.denx.de/wiki/U-Boot>. [32](#)

**UART** universal asynchronous receiver/transmitter, which is used to exchange data between two systems using two independent serial lines. [12–15](#), [26](#), [33](#), [35](#), [36](#), [51](#)

**UMTS** Universal Mobile Telecommunications System. [28](#), [44](#)

**USB** Universal Serial Bus. [12](#), [13](#), [15](#), [20](#), [26](#), [27](#), [30](#), [51](#), [52](#)

**USSE** unified scalable shader engine. [53](#)

**V4L2** Video for Linux 2. [33](#), [38](#), [53](#)

**WDT** timer for recovering from malfunction by triggering a hardware reset. [26](#)

**WLAN** Wireless Local Area Network. [7](#), [8](#)

**X-Server** Windowing System for bitmap based displays, that is usually used on UNIX and Linux systems.  
[25](#)