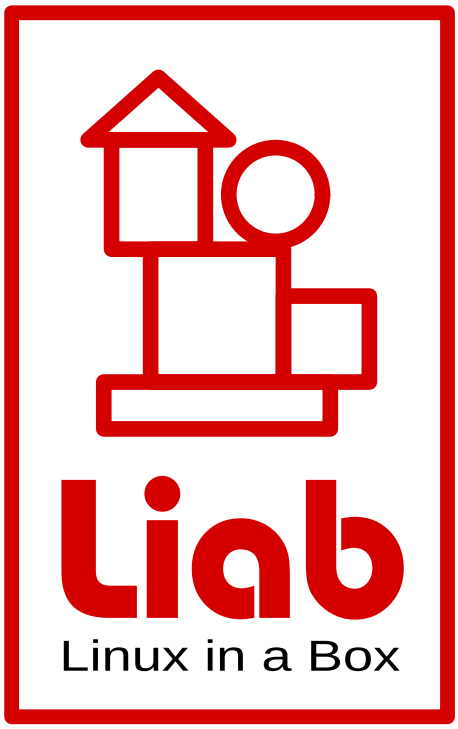


User's Manual for the

nanoLIAB

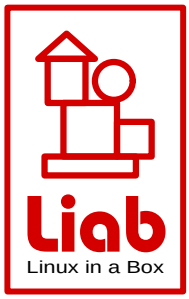
microprocessor board

Version 00.02, November 2006

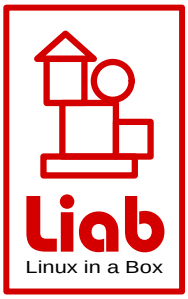
This document describes the use of the nanoLIAB microprocessor board. An introduction to the LIAB (Linux In A Box) concept is given together with a description on how to get the microprocessor board up and running. Procedures for the development of software and hardware extensions are also given.

Please also consult the accompanying nanoLIAB CD-ROM for software distribution and schematics. Note that all parts of the nanoLIAB hardware and LIAB bootloader is copyright of LIAB ApS.

LIAB ApS
Østre Allé 6
DK-9530 Støvring
Tlf: +45 98 37 06 44
mail: info@liab.dk
http://www.liab.dk



NOTICE:



The information in this document is subject to change without notice. Further, the software and documentation are provided "as is" without warranty of any kind including, without limitation, any warranty of merchantability or fitness for a particular purpose. Even further, LIAB ApS does not guarantee or make any representations regarding use or the result of the use of the software, hardware or written material in terms of correctness, accuracy, reliability or otherwise.

Contents

0. Editorial Notes (read this first)

This document describes a small but yet powerful computer system called the "nanoLIAB".

For the impatient user: If you are eager to experience the features of the Linux system on the nanoLIAB board, plug it into your computer as described in chapter 3.

Notational conventions: Throughout the manual, it is assumed that a host PC running the Linux operating system is used for the communication with the nanoLIAB system. Screen dumps and examples of human interaction are printed using fixed-spaced typewriter letters:

```
POSIX conformance testing by UNIFIX
Page-cache hash table entries: 16384 (order: 4, 65536 bytes)
CPU: Testing write buffer: pass
Linux NET4.0 for Linux 2.6
...
```

The following two prompts signify that the user is interacting with the host PC directly, either as normal user ("`user@host$`") or as superuser ("`user@host#`"):

```
user@host$ ls
< files on the host PC >
user@host$ su
Password: < enter root password for the host PC>
user@host#
...
user@host# < press ctrl-D >
user@host$
...
```

Using either a serial communication program like "`cu`" or a network terminal program like "`telnet`", you may communicate with the nanoLIAB system. To signify this, the prompt "`root@liab#`" will be used:

```
user@host$ telnet liab
```

```
Trying 192.168.1.180...
Connected to liab.
....
liab login: root
password: < enter root password for the LIABARM9200 >
root@liab# ls
< files in /root directory of the LIABARM9200 >
...
```

The root password for the nanoLIAB board can be found in the covering letter.

1. Introduction

The Linux In A Box (LIAB for short) project was started in the summer of 1998 at the Institute of Electromagnetic Systems at the Technical University of Denmark, DTU. The aim was to develop a small microprocessor platform feasible of performing control and data acquisition task in relation to an antenna measurement facility. A prototype and the first generation of the LIAB board were developed at DTU.

In the fall of 2000, all activities were moved to the Danish company "LIAB ApS", a company focused on developing single board computers running the Linux operating system.

All parts of the hardware and bootloader software are copyright of LIAB ApS. However, the hardware and most of the software are open-sourced. For the hardware this means that everybody gains full insight in all schematics and PCB designs. Similarly, everybody have full insight in the patches for the Linux kernel and images for RAM disks. You are free to distribute the full documentation of the hardware and source codes of software, as long as you do not make changes to either parts. However, you may distribute changes and contributions to the LIAB project, but you must clearly mark which parts are yours and which are part of the distributions from LIAB ApS. If you sell a product that uses the LIAB bootloader or if you develop a product using the bootloader for use by, or on behalf of a commercial entity, LIAB are entitled to a royalty fee. Additionally, LIAB should also be compensated if products using the bootloader is treated as proprietary, thus enabling a competitive advantage to a company. Please contact LIAB ApS for more details.

1.1 The Concept

During the conceptual phase of the development of microprocessor-based control systems it is often recognized that the task of developing software takes up a major part of the total time needed. A mean to reduce the extent of the software task is to use an operating system (OS). Choosing the open-sourced operating system Linux for a project will not only keep the basic cost of the software at a reasonable level (that is, no cost at all!), but the software development process

will also benefit from the vast amount of applications written for Linux. Due to its widespread use, drivers for all sorts of hardware can be found on the Internet and the programming environment is well documented, both in books, [1] [2] [3] [4] [5] , but also in numerous `README-`, `FAQ-` (Frequently Asked Questions) and `HOWTO-`files on the Internet. On top of that, programmers with experience in the UNIX operating system may easily migrate to the Linux, since in fact Linux is yet another clone of the UNIX OS. In particular, classical textbooks on UNIX, [6] [7] [8] [9] apply almost directly to Linux.

The Linux project was started in 1991 by Linus Torvals and for a long period only the Intel i386 processor architecture was supported. However, Reduced Instruction Set Computer (RISC) processors have gained considerable use, an efforts have been made to port Linux to such processors.

The ARM (Advanced RISC Machine) processors are widely used in mobile phones since these posses high performance, low power consumption and low cost. High end ARM processors (`ARM72x`, `ARM92x`) contain a memory management units (MMU) together with cache systems. Such processors are well suited for the Linux operating system. Specifically, the Atmel AT91RM9200 microprocessor containing a ARM920 core has been used to create the nanoLIAB series microprocessor boards.

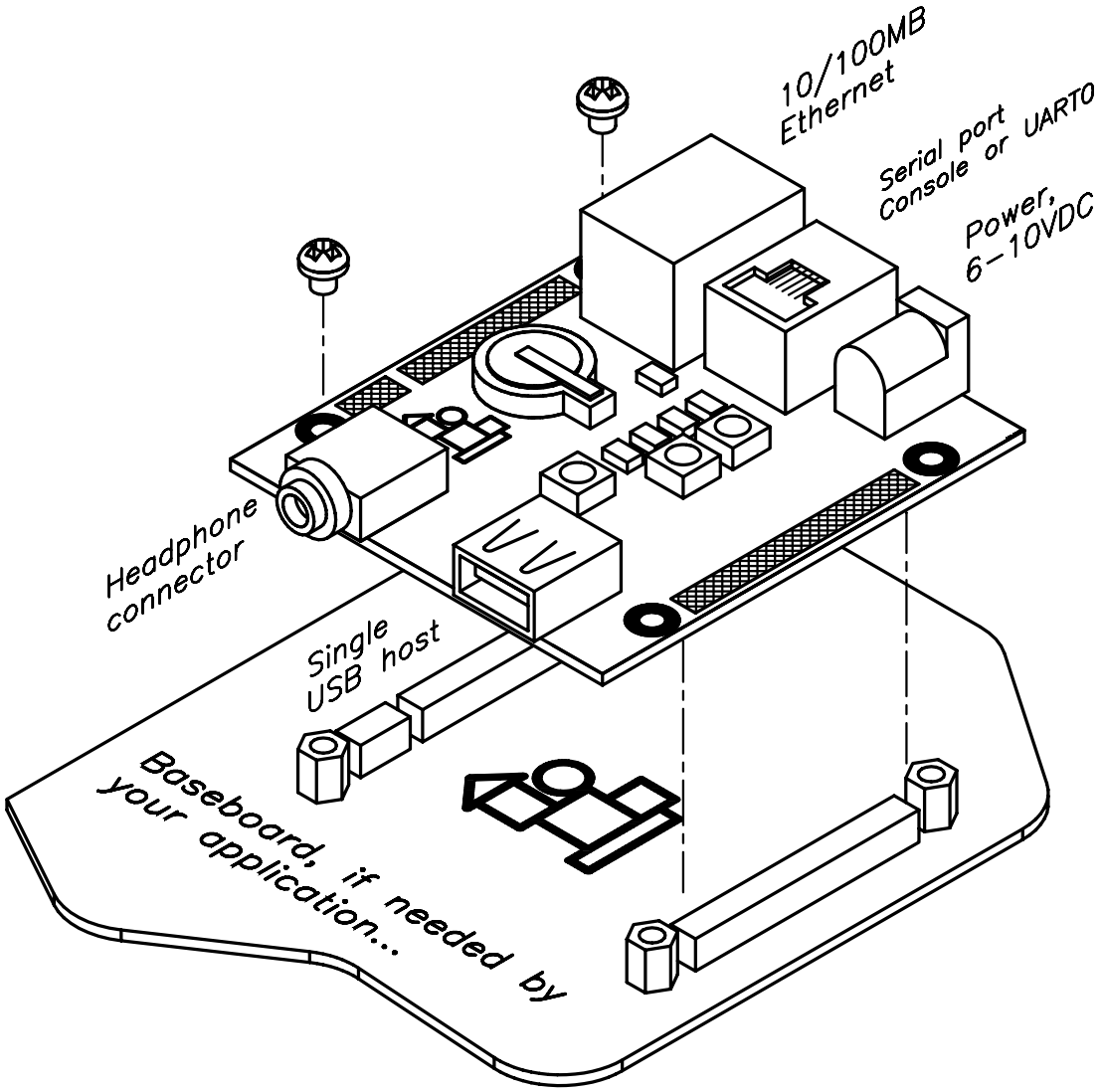
Based on the AT91RM9200, the nanoLIAB microprocessor board employs a hardware structure which allows fast system prototyping and a variety of custom interfacing possibilities. The board is a self contained, fully functional Single Board Computer (SBC) with three interfacing connectors in form of pin-headers. This allows the main microprocessor board to be mounted on different baseboards with various interfacing and on-board features. With reference to Fig. 1.1 the nanoLIAB microprocessor board is identified as the top board.

Currently, only a bread board for experimental use is available from LIAB. However, it is expected that more application specific baseboards will follow. These could include a multimedia baseboard, featuring graphics and camera systems, and a baseboard for industrial controls equipped with relays and digital and analog I/O interfaces.

Please consult LIAB ApS if you like to engage in a discussion on the design, manufacturing and testing of a baseboard, which can fulfill your specific requirements.

The LIAB board is presently distributed with a version 2.6.16 Linux kernel and a Linux file-tree which is an extract of the Debian Linux distribution. The shared libraries in this file-tree are based on a recent version of the GNU libc library: libc6.

Please note that the ARM9 core is not compatible with x86 core of you Personal Computer! Programs for the nanoLIAB must be compiled for the ARM platform using a cross-compiler, e.g. the one included on the accompanying CD-ROM.



Figur 1.1: *The nanoLIAB microprocessor board. Connections to a possible baseboard is shown as rectangular boxes on the baseboard.*

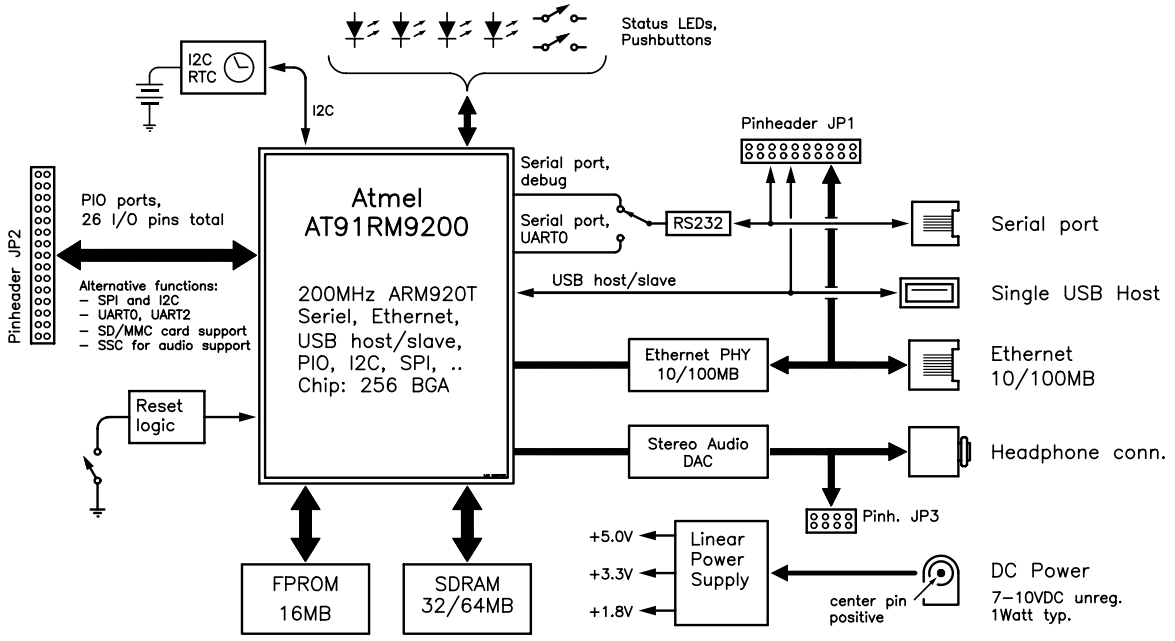
2. The nanoLIAB Hardware

The Linux In A Box (LIAB) ARM-based solution provides an excellent platform for small control and data acquisition systems that needs to be supervised over the Internet. The microprocessor board uses one of the most powerful Atmel ARM based microprocessor currently available, providing both a number of built-in peripherals, such as USB host and Ethernet, and also features very low power consumption (0.8-1.2 Watts). The on-board FLASH PROM memory provide storage for the bootloader, the Linux operating system and application software and data. The nanoLIAB board is equipped with standard connectors and interfaces, as well as pin headers for interconnection with other pieces of PCB, e.g. a baseboard. However, the nanoLIAB board is self contained, as you may connect directly to the nanoLIAB using an Ethernet patch cable, a Serial cable and a connection a USB host plug.

In the following, the features of the nanoLIAB microprocessor board are described together with a discussion of the signals in the three pin headers located on the bottom side of the board.

2.1 The nanoLIAB Microprocessor Board

The microprocessor board is an 6-layer PCB (Printed Circuit Board) which contains the Atmel AT91RM9200 microprocessor, FPROM and DRAM memory, peripheral components and a linear power supply. To provide flexibility, most accesses to the features of the board can be done through either the standard connectors or the three pin-headers located on the bottom side of the board. A block diagram of the nanoLIAB microprocessor board is shown in Fig. 2.1.



Figur 2.1: *Block diagram of the nanoLIAB microprocessor board.*

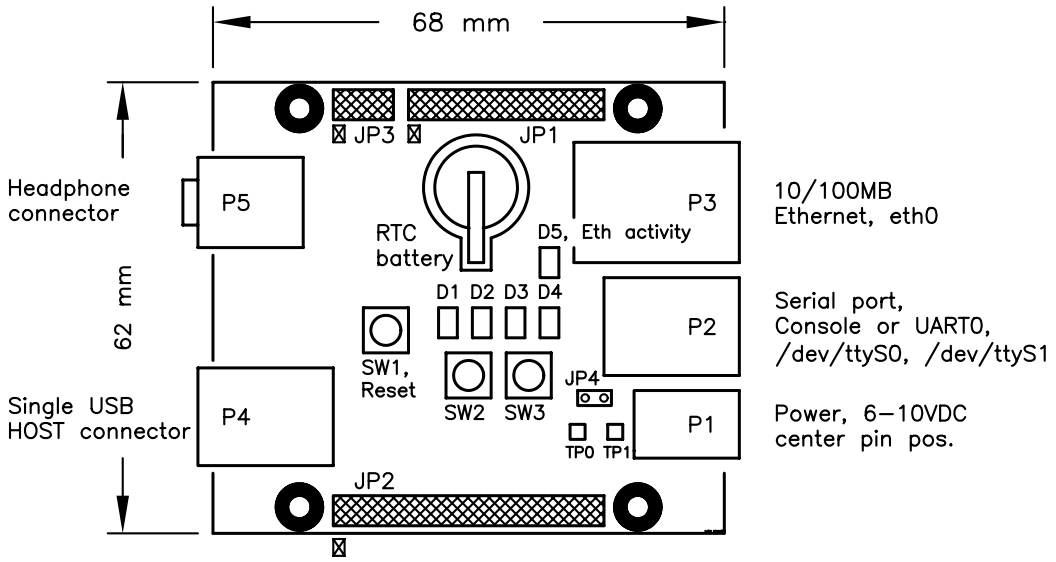
The nanoLIAB microprocessor board has the following features:

- Atmel AT91RM9200 ARM microprocessor (small outline BGA package) running at 180 MHz. The microprocessor includes interrupt and DMA controllers, serial channels, timers, a real time clock, a FLASH and SDRAM interface and lots of general purpose digital I/O ports. In addition, several dedicated interfaces are present: Serial ports, Ethernet, USB host, USB slave, I^2C, SPI, ...
- 16 MB non-volatile FLASH PROM memory for the boot loader the Linux kernel (presently version 2.6.16) and ramdisk containing the root file system.
- 32 MB (64MB optionally) synchronous DRAM functioning main work storage for the Linux system.
- An asynchronous serial ports multiplexed between two UARTs: the debug UART and UART0. Using the debug UART you may watch the console output from the Linux system. Using UART0, you have at hand a general purpose UART for login, modems etc. The multiplexed port employs a RS232 line driver, giving access to four signals: `TXD`, `RXD`, `CTS` and `RTS`.
- An 10/100 Mbit Ethernet interface, complete with an Ethernet PHY. You may connect the a patch cable to the RJ45 socket on the nanoLIAB board for network access to the nanoLIAB.

- An USB host controller with a two port root hub. Thus, two USB devices can be directly connected to the microprocessor board at the same time. One of the ports are accessible through the standard USB connector on the board.
- Real Time Clock (RTC) with battery backup.
- Operator interface having four LEDs and two push-buttons. A third push-button is provided to reset the computer.
- A high performance stereo audio DAC, feasible of producing CD quality audio signals. A headphone amplifier is also included, capable of producing 2×30mW into 32 Ω speakers.
- A linear supply producing +5V, +3.3V and +1.8V for the on-board logic.
- Three pin-headers for power, digital IO and audio output.

2.2 Board Layout

The electronics for the microprocessor board is assembled on a PCB using four layers for signals and two for power. The dimensions of the PCB are 62×68mm^2 and the weight is less than 50 grams. Nearly all components are placed on one side of the PCB, except for the connectors, LEDs, push-buttons and battery holder for the RTC battery.



Figur 2.2: *Layout of connectors, LEDs, switches and pin-headers on the nanoLIAB microprocessor board.*

The layout of connectors, pin-headers, jumpers and other large components is shown in Fig. 2.2. The three pin-headers JP1 to JP3 are show as hatched rectangles on the figure. The purpose of the individual headers JP1-3 is described in Table 2.1 on page 14.

2.2.1 Power Supply

The microprocessor board must be powered by a DC supply having a voltage between 6 and 12 volts using the power jack connector P1. The center pin in the jack must be positive. The board consumes around one watt and it has been tested to work in temperatures ranging from -10^{o}C to +60^{o}C. Power can also be induced using the pin-header JP1. Similarly, regulated voltages of +5 Volt and +3.3 Volt can be accessed using pins in JP1 and JP2.

2.2.2 Standard Connectors P2-P5

Using the standard connectors P2-P5, you may access the individual parts of the microprocessor board:

- Connector P2: Standard RJ12 (Modular 6P6C) connector for serial communication. See Fig. 3.2 on page 14 for the pin-out of the P2 connector. As a default, you connect to the serial debug console (the DBGU UART on the microprocessor, /dev/ttyS0), when accesing the P2. Thus, you can configure the nanoLIAB using the bootloader, watch the boot process and finally log into the system using registered user-ids and passwords. The default communication parameteres are: 115200 baud, 8N1. However, using the program nanoctrl you may switch the serial multiplexer on the nanoLIAB to facilitate communication with UART0 (/dev/ttyS1). The pins in P2 are also found on pin-header JP1.

- Connector P3: Standart RJ45 connector for Ethernet. Both 10 and 100 Mbit/sec is supported and in presence of link pulses on the connected Ethernet cable, LED D5 is lit. Activity on the net is signified by blinks. You *may* access the Ethernet interface using pins in pin-header JP1. However, you cannot just connect yet another RJ45 connector of the same type (Pulse Engineering J0026D01) in parallel with that on the nanoLIAB board. This is due to the fact that the connector in question includes small signal transformes. Thus, you have to unmount the connector on the nanoLIAB, if you want to use an connector external to the nanoLIAB board.

- Connector P4: Standard single USB HOST connector for e.g. memory sticks, WEB cameras, printers, etc. The nanoLIAB includes a two USB HOST and one USB slave interface. All three interfaces are accessible on pin-header JP1.

- Connector P5: Standard stereo headphone connector for the high quality stereo DAC on the nanoLIAB board. The headphone outputs and other signals into and out of the stereo DAC are present on pin-header JP3.

2.2.3 LEDs and Switches

The nanoLIAB microprocessor board is equipped with 5 LEDs (D1-5), three push buttons (SW1-3). The first four LEDs are fully programmable, and can be used as an operator interface for debugging and configuration purposes. the fifth LED indicates the status of the network interface: link (constant light) and activity (blinking). SW1 acts as a reset push-button, whereas the state of SW2-3 can be read from software. In section 4.7 a Linux kernel module for the control of these elements are presented, together with some programming examples. However, you may also use the program nanoctrl, described in section 4.6 on page 27, to control the LEDs and read the switches.

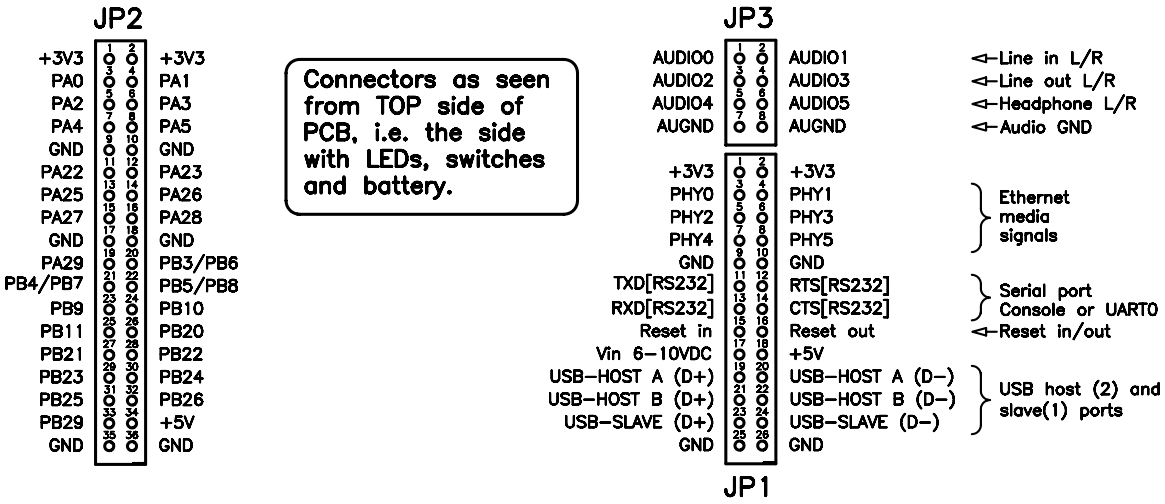
2.2.4 Pin Headers JP1 to JP3

The three pin headers present on the bottom side of the nanoLIAB microprocessor board are labeled JP1, JP2 and JP3. Their location can be observed on Fig. 2.2 and pin no. 1 is marked with a crossed rectangle. All pin headers consist of two rows of gold-plated pins placed on a 2.00mm module grid. The purpose of the individual pin headers is described in Table 2.1. The pin

Item	Pins	Purpose
JP1	26	Power and communication: • Power in: 6-12 Volts DC, unregulated • Regulated power out: +5V and +3.3V • Ethernet • Serial port (DBGU or UART0) • USB HOST (×2) and USB slave ports • Reset in/out
JP2	36	Peripheral pins: • 27 Programmed Input-Output (PIO) All pins have three functions: general IO and two special functions (timers, SPI, I^2C, serial ports, SD/MMC card support, interrupt, ...). You should consult the User's Manual of the AT91RM9200 microprocessor for further descriptions.
JP3	8	Audio connector: • Line in/out and headphone out

Tabel 2.1: *Pin-headers on the nanoLIAB microprocessor board: JP1 to JP3.*

headers are feasible for mating PCBs or ribbon cable connectors. You may consult the schematics, found on the accompanying CD-ROM in the directory `hardware/processorboard/`, for the specific functions of the individual pins of the pin headers. However, the functions of the pins in the three connectors, JP1, JP2, and JP3, are given in Fig. 2.3.



Figur 2.3: *Signals of the three connectors on the nanoLIAB,* `JP1-3`.

3. Get your Board Up and Running

When delivered from LIAB ApS, your nanoLIAB microprocessor board is preloaded with a boot loader and a Linux system. This system will boot when power is connected using connector P2.

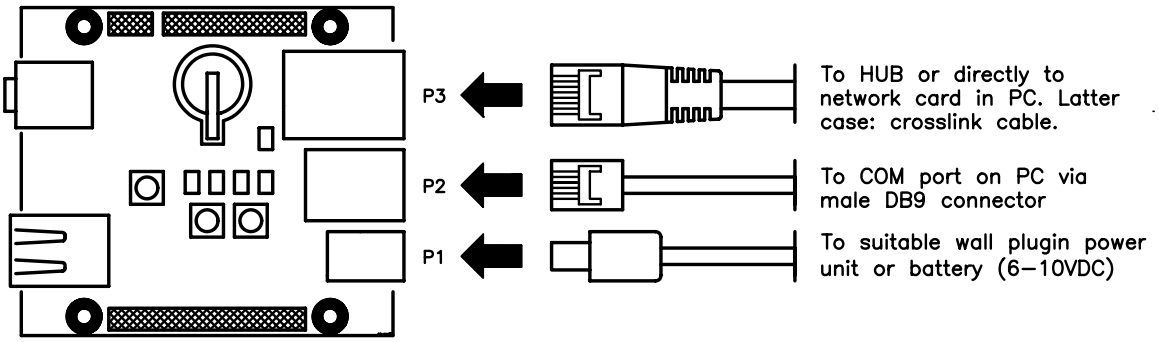
PLEASE NOTE: Electrostatic discharges (ESD) can damage your LIAB board and care must be taken to avoid them. You should wear a grounded anti-static wrist strap before unpacking the nanoLIAB from the protective, anti-static bags it was delivered in. In addition, the nanoLIAB should be kept on a grounded, static-free surface.

3.1 Required Items

You need the following items to begin using the LIAB board:

- One nanoLIAB microprocessor board.
- A serial cable. The end to be connected to the nanoLIAB must be equipped with a RJ12 (6P6C) plug. An appropriate connector for your host computer must be located in the opposite end of the cable. Wiring of a suitable cable is shown in Fig. 3.2.
- A power source. A DC power supply with a voltage in the range 6 to 12 volts, capable of delivering at least 2 watts, is required. A simple wall plug-in power module with adequate power rating is usable.
- A host computer with a serial channel and software for a terminal emulator. The first login into the nanoLIAB can be done using the serial line. Next, you can use the serial line to configure boot parameters such as IP number, subnet mask, etc.

- A twisted pair Ethernet cable is needed when you have configured the board with the network parameters. If you want to connect the nanoLIAB to a Ethernet hub or switch, any standard cable will do. Alternatively, the LIAB can be connected directly to a host computer using a cross-link cable.



Figur 3.1: *Connections to the exterior world: power, serial port and network.*

3.2 Unpacking and Serial Connection

A suitable serial cable should be included with the nanoLIAB board when delivered from LIAB ApS. Alternatively, you can make one yourself if you have at hand one RJ12 (Modular 6P6C) plug, a length of flat telephone cable with six wires and a female DB-9 connectors. The wiring you need to make is shown in Fig. 3.2

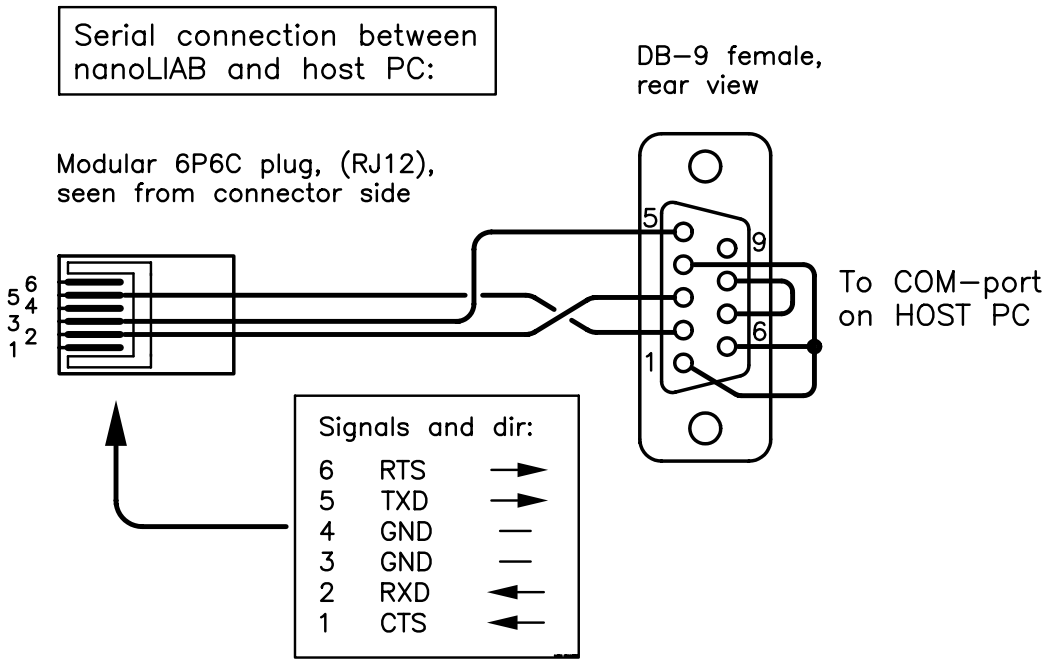
3.3 Start a Terminal Emulator and Apply Power!

Having connected the LIAB board to your host computer using the serial cable you are ready to apply power through power connector `P1`. A terminal emulator on the host computer must be started and configured for **115200 baud, 8 databits and no parity bit**. On a PC running Linux you may use the terminal program "`cu`" as described in Appendix A. Having started "`cu`" at 115200 baud and now applying power to the LIAB, you will see a boot sequence like this:

```
user@host$ cu -l /dev/ttyS0 -s 115200 Connected.
Boot>

------------------o LIAB Bootloader o----------------

Release: 1.0, November 20, 2006 at 15:01 by msa
Copyright LIAB ApS.
```



Figur 3.2: *Serial communication for the nanoLIAB: connectors and wiring diagram.*

```
The bootloader will now search for a compressed kernel and file-
system and try to boot up a Linux system. IF YOU WANT TO GET INTO
THE BOOT LOADER, YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: **...
```

The asterisks ("*") at the end of the last line are time indicators, each separated by a one second interval. If no user intervention occurs within five seconds, the bootloader will try to locate a Linux system and boot it. You may try this out and consequently you will see a boot-up sequence like this:

```
--------------------o LIAB Bootloader o-----------------

Release: 1.0, November 27, 2006 at 15:01 by msa
Copyright LIAB ApS.
The bootloader will now search for a compressed kernel and file-
system and try to boot up a Linux system. IF YOU WANT TO GET INTO
THE BOOT LOADER, YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: *****
PHY reset completed OK
Scanning FPROM memory range 0x00000000 to 0x00ffffff
for gzipped kernel and initrd images:
GZIP image no. 1 found at addr 0x00020000
Filename ..........: vmlinux.bin
Comment ...........: <no comment>
Timestamp .........: Nov 28 09:46:25 2006 UTC
GZIP image no. 2 found at addr 0x001a0000
Filename ..........: initrd
```

```
Comment ...........: <no comment>
Timestamp .........: Nov 28 10:55:58 2006 UTC
Loading kernel at 0x20008000
-- Now Decompressing Image! ---- Now Decompressing Image! ---- Nodecompressed
age 2133904 bytes
Bootloader: now putting Linux boot tags at 20000100
Starting Linux kernel ...
Linux version 2.6.16 (msa@msa) (gcc version 3.3.2) #31 PREEMPT Mon Nov 27 15:3
30 CET 2006
CPU: ARM920Tid(wb) [41129200] revision 0 (ARMv4T)
Machine: Atmel AT91RM9200-DK
Memory policy: ECC disabled, Data cache writeback
Clocks: CPU 165 MHz, master 55 MHz, main 14.745 MHz
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists
Kernel command line: liabETH=00:90:82:FF:03:F0 liabIP=192.168.1.153,8,192.168.
1 liabHOST=msa3.liab.dk liabJFFS2=/jffs2 liabRUN=/jffs2/StartApplication
...
...
< many lines of Linux kernel initialization messages >
...
Starting httpd...
Executing file /jffs2/StartApplication
Setting system clock
Copying '/jffs2/root/*' to '/'
Loading nanoLIAB kernel module

LIAB distribution 6I
LIAB ApS, visit http://www.liab.dk

liab login:
```

After approximately 10 seconds, you will get the Linux login prompt where you can login as users "`root`" or "`liab`". The passwords are supplied on a separate covering letter in the shipment from LIAB ApS.

3.4 Network Configuration: Send Three Dots

In section 3.3 it was suggested that no user intervention was taken during the first five seconds after power was applied. As a consequence, a Linux system was booted. This time we want to get into the bootloader menus in order to configure network parameters. Press the reset button SW1 on the nanoLIAB standard baseboard and then immediately send three dots, "`...`", to the LIAB from your terminal emulator. Now you will get a bootloader prompt:

```
THE BOOT LOADER, YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: ***
LIAB bootloader, 'h' for help
Boot>
```

Try the "h" command to get a list of possible commands:

```
Boot>h
 d <start> <end>  : Display memory from <start> to <end>
*f                : Go into the FLASH PROM utility submenu
 h                : Help (this text)
 j <addr>         : Jump to <addr>, default 0x21000000
 l                : Load images using uuencoded data
*p                : Display and edit boot and network parameters (a la lilo)
 q                : Quit monitor and continue boot procedure
 r <rate>         : Set baudrate (0:9600, 1:19200, 2:34800 3:57600 4:115200)
 s                : Scan memory for GZIP images
items marked with "*" give access to submenues.
Boot>
```

As described, your LIAB will be equipped with a bootloader, a Linux kernel image and a disk image when shipped from LIAB ApS. The kernel contains a driver for the network hardware on the LIAB board. In addition, the root file system in the disk image contains scripts for initializing the network system with IP-number, subnet mask, default gateway, etc. At the end of the boot procedure, both a network daemon, `xinetd`, and a web server daemon, `httpd`, is started. Thus, you may connect to your LIAB board using e.g `telnet` and look at its webpages using a browser. The only thing you need to setup is the basic network parameters which is done using the boot loader. From the "`Boot>`"-prompt, enter the parameter sub-menu by typing the "p"-command. Help on the sub-menu can be obtained using the "h"-command:

```
Boot>p
Entering the boot/network parameter editor, 'h' for help,
use options 'n' and 'a' for editing of network stuff.
Param>h
 a                : alter the network specifications
 h                : help (this text)
 d <line>         : delete the specified parameter-line
 i <line> <param> : insert text <param> before parameter-line <line
                    NOTE: the text string must NOT contain spaces!
 n                : show the network specifications
 p                : print the boot/network parameters
 q                : quit boot/network parameter submenu
 w                : write new parameters back to FPROM
Param>
```

Now you are ready to enter your IP-number, subnet mask, default gateway, host- and domain-name and domain name server to the LIAB using the "a"-command.

You are asked the following:

1. If you want to enter the IP network specifications, answer "y" for yes and enter the IP-number and subnet mask for the primary Ethernet connection

(eth0). Review the settings and if correct, type "n" when you are asked if you want to further change the IP settings.

2. If you want to enter the host/domain specifications, answer "y" and enter the host-name, the domain-name and, if applicable, the IP-number of a domain name server. Review the settings and if correct, type "n" when you are asked if you want to make further changes.

Please note that you need to supply a domain-name in order to make the web server work properly. You may use the domain-name "dummy" in case you are on a local net without any nameserver.

Having entered the network parameters, you may first review them using the "p"-command and then write them back to the FPROM using the "w"-command:

```
Param>p
 1: liabIP=192.168.1.182,8,192.168.1.1
 2: liabHOST=liab2.liab.dk
Param>w
Do you want to write the parameters back to FPROM? [y/n]>y
Boot parameters start at: 0x00fff800
Param>
```

The two parameters: "liabIP=..." and "liabHOST=..." are passed to the Linux kernel just before it boots. This process is in nature identical to the passing of boot parameters from the LILO-prompt to the Linux kernel on a PC.

You are now ready to boot the Linux system. Either, you press the reset button SW2 on the nanoLIAB standard baseboard or you enter two successive "q" (quit) commands:

```
Param>q
Boot>q
Scanning FPROM memory range 0x00000000 to 0x00ffffff
...
...
Loading kernel at 0x20008000
-- Now Decompressing Image! ---- Now Decomp...
decompressed image 2133904 byte
Bootloader: now putting Linux boot tags at 20000100
Starting Linux kernel ...
...
```

You will now observe a boot sequence similar to the one printed on page 18. Eventually, you will get a login prompt where you can login as users "root" or "liab". You may also connect to the LIAB using "telnet" and you should consider trying to browse the homepage on the LIAB using e.g. "firefox" or any other web browser.

4. The LIAB Distribution

A CD-ROM is enclosed in the shipment from LIAB ApS. It contains documentation and software for the nanoLIAB board. The distribution is open-sourced as described in section 1. This means that you are free to redistribute it.

4.1 Installing the Distribution

Before installing the distribution from the CD-ROM you have to make sure, that you have at least 640 MB of free space on the hard drive of your Linux PC-compatible computer. To install the nanoLIAB distribution you must get root access and unpack the tar-file "`<distribution name>.tgz`" into a suitable directory using something like:

```
user@host# tar -xvz -C /<MyHomeDir> -f liab4I-ARM-sep-2005.tgz
```

Files in the distribution are either owned by "root/root" (uid:1, gid:1) or "liab/users" (uid:998, gid:100). If convenient, you might change the ownership of the files in your local copy of the distributions provided that you do not change the ownership of the files in the directory `/liabdisc`. To change the ownership of the files to e.g. "randi:users", get root access and enter the distribution directory. Then type:

```
user@host# find . -user 998 -exec chown randi:users {} \;
```

4.2 Installing the Cross Compiler

To compile programs for the ARM architecture on a standard x86-based PC, a cross compiler is needed. The directory `software/crosscompiler` contains not only a precompiled cross compiler, but also scripts to build the cross compiler from sources found on the Internet.

To install the cross compiler all that is needed is to copy the "`opt/`"-directory from the CD-ROM to the root ("`/`") on your Linux computer as root:

```
user@host# cd /mnt/cdrom
user@host# cp -a opt /
```

The normal way of using the cross compiler is prepending the path

```
/opt/crosstool/arm-softfloat-linux-gnu/gcc-3.3.2-glibc-2.3.2/bin
```

to your search path by inserting something like this in your profile (`.profile`, `.bash_rc`, `.tcshrc`, ...):

```
...
PATH=$PATH:/opt/crosstool/arm-softfloat-linux-gnu/ \
gcc-3.3.2-glibc-2.3.2/bin
...
```

Now you can perform cross compilations using `arm-softfloat-linux-gnu-gcc` instead of `gcc`, `arm-softfloat-linux-gnu-c++` instead of `c++` and so on.

To instruct a Makefile to use `arm-softfloat-linux-gnu-gcc` instead of `gcc`, you simply add the line "`CC = arm-softfloat-linux-gnu-gcc`" to the top of the Makefile, or adding it on the command line like this:

```
user@host$ make myprogram CC=arm-softfloat-linux-gnu-gcc
```

See the directory `software/crosscompiler/hello_arm` for a Makefile example.

4.3 Contents of the Distribution

This description of the distribution pertain to the nanoLIAB file tree as installed on your host PC using the installation procedure described in section 4.1.

The directory "`/hardware`":
This directory contains hardware documentation for the LIAB boards. Documentation of the current versions of the nanoLIAB microprocessor board and the standard baseboard as supplied in the evaluation kit are placed here. The documentation contains schematics (diagrams) and component layouts for both boards.

The sub-directory "`/hardware/processorboard`":
Schematics and component layout for the nanoLIAB microprocessor board.

The directory "`/software`":
This directory contains software for the nanoLIAB microprocessor board: bootloader, Linux kernel and file trees for disk images to be loaded into the FLASH PROM.

The sub-directory "`/software/liabboot`":
Binaries for the bootloader for the nanoLIAB microprocessor board. This directory also contains various shell scripts relevant for download of a new bootloader, a new kernel and disk images.

The sub-directory "`/software/liabkernel`":
Modified kernel source of the version 2.6.16 Linux kernel which is able to boot on the nanoLIAB microprocessor board. The directory also contains a compressed tar-image of the original kernel source together with patch-files.

The sub-directory "`/software/liabdisc`":
File tree for a fully operational Linux system with scripts for boot-up and the most relevant commands. Contains: network, `bash`, `vi`, shared libs, /dev, /proc, Apache httpd, etc. Based on Debian version 6.2. Scripts to create a compressed disk image suitable for download to the FLASH PROM on the liab.

The sub-directory "`/software/crosscompiler`":
The crosstool build script for gcc used to build the cross compiler. The directory also contain a precompiled cross compiler for Atmel AT91RM9200 ARM microprocessor. For instructions on how to install and use the cross compiler, see 4.1.

The directory "`/pdf`":
Documentation of the various chips used on the nanoLIAB microprocessor board and nanoLIAB standard baseboard. The directory "`/pdf/atmel`" contains full documentation of the AT91RM9200 ARM chip.

4.4 Demo program for the LIAB

This section provides a short introduction on the use of a Linux PC as a development platform for writing application for the nanoLIAB system. Before you start this introduction, make sure that you have entered the network parameters into your LIAB as described in section 3.4 and that you can get in contact with your nanoLIAB using the network (test the connection using eg. the "`ping`"-command).

After installing the LIAB distribution, including the cross compiler, on your Linux PC as described in section 4.1 you will find an example of a very simple Linux application program in the directory
`/software/crosscompiler/hello_arm`. To get into this directory, type:

```
user@host$ cd <your liabarm dist. path>/software/crosscompiler/hello_arm
```

In the /hello_arm directory you will find the source file hello_arm.c for a small program that does nothing else than print out a short text when executed. To compile the hello_arm.c file using the ARM cross compiler, type:

```
user@host$ arm-softfloat-linux-gnu-gcc hello_arm.c -o hello_arm
```

or alternatively, use the make utility:

```
user@host$ make hello_arm
```

The output file: hello_arm is an executable that can run directly on the nanoLIAB.

First, boot up your LIAB and use the program ftp to transfer the executable hello_arm to the nanoLIAB target:

```
user@host$ ftp <IP of the LIAB-board, typically 192.168.1.180>
Connected to liab (192.168.1.180).
220 Welcome to LIAB FTP service.
Name (host:user): root
331 Please specify the password.
Password: < enter LIABARM9200 root password >
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put hello_arm
226 File receive OK.
ftp> bye
221 Goodbye.
user@host$
```

Next, connect to your nanoLIAB system from your Linux PC using e.g. cu as described in Appendix A. Login as root and make the file hello_arm executable

```
user@host$ cu -l /dev/ttyS0 -s 115200
connected
root@liab# chmod +x hello_arm
root@liab# ./hello_arm
Hello ARM World
root@liab#
```

As an alternative to ftp, you might use the command rcp (remote copy) to transfer files between your host PC and the LIAB board. However, before you can do that you must log onto the LIAB and add an entry to the .rhosts-file in the directory /root:

```
user@host$ telnet <IP of the LIAB-board, typically 192.168.1.180>
Connected to liab.
...
liab login: root
password: < enter LIABARM9200 root password >
root@liab# cat >> .rhosts
< IP-number of your host PC > < your userid at the host PC >
< presse Ctrl-D >
root@liab#
```

To transfer the file `hello` to the LIAB use the `rcp`-command on the Linux PC:

```
user@host$ rcp hello_arm root@<LIAB's IP-number>:.
```

The file `hello_arm` is now copied into the home directory of user `root`. To run the program, switch to the telnet-session stated above and type "`./hello_arm`":

```
root@liab# ./hello_arm
Hello ARM World
root@liab#
```

4.5 Loading the nanoLIAB Module

Kernel modules in Linux are units of compiled code that can be loaded and unloaded from the kernel on demand. One type of kernel module is device drivers, allowing the system to communicate with connected pieces of hardware, e.g. serial ports, printers, etc.

The preloaded Linux system that came with your nanoLIAB microprocessor board includes a device driver for reading and controlling the state of the buttons and LED's located on the nanoLIAB microprocessor board.

The following assumes that you have created either a telnet connection or a a serial connection using "`cu`" as described in appendix A to the nanoLIAB board.

To use a Linux kernel module it must first be loaded into the kernel. If any communication to and from the module is necessary a corresponding device-special file must also exist in "`/dev`". To load the module the "`insmod`" program is used, and the device-special file is created using "`mknod`". In the example below the module is first inserted, and followingly a device-special file, called "`nanomod`" is created.

```
root@liab# insmod nanomod
root@liab# mknod /dev/nanomod c 63 0
```

Through the device-special file you now have read and write access to the buttons and LEDs. A thorough description of the possible interactions with the module "nanomod" can be found in a "README"-file on the CD-ROM at:

```
software/liabkernel/liab-modules/liabarmmod/kernelspace
```

For now, we just want to light up the ten LEDs and the individual segment in the seven segment display. Try the following series of commands:

```
root@liab# echo "^L0" > /dev/nanomod    < light LED D1 >
root@liab# echo "^L1" > /dev/nanomod    < light LED D2 >
root@liab# echo "^B1" > /dev/nanomod    < blink LED D2 >
root@liab# echo "^D1" > /dev/nanomod    < darken LED D2 >
root@liab# echo "^V7" > /dev/nanomod    < Set leds D4..D1 = 0111 >
```

It is also possible to read the state of both push buttons (SW2-3) through the "nanomod" module. To read the state of the two push-buttons, we type:

```
root@liab# dd if=/dev/nanomod count=1
00
```

The two digits returned represents SW1 and SW2 (1 when button is depressed, 0 if not). The source code for the module and an example of how to use the module within software is included on the accompanying CD-ROM. See the directory

```
/software/liabkernel/liab-modules/liabarmmod/.
```

4.6 The Board Control Program: nanoctrl

Instead of accessing the kernel module directly, LIAB has written a small program to:

1. Control the four LEDs D0-D3
2. Read the status of the two switches SW2-3
3. Control the serial port multiplexer which chooses whether to out the debug UART, DBGU, or UART0 on the RJ12 connector P2. (default at boot up is always debug UART)

The program actually uses the kernel modules nanomod and the device special file /dev/nanomod. Thus, the module has to be loaded and the device special file must exist. However, the default start-up application located in FLASH disk as /jffs2/StartApplication does this.

You may see the options for the nanoctrl program by entering:

```
root@liab# nanoctrl -h
Program to operate the various facilities of the nanomod module.
  Usage: ..# nanoctrl <options>
Options:
  -h  --help                : Help (this text)
  -r  --read                : Read button states
  -l  --led [VALUE]         : Set LED states to VALUE, where
                              VALUE is a hex number
  -s  --set-serial [MODE]   : Set the P2 serial port into AUX
                              serial or DEBUG serial mode. MODE
                              can be one of either AUX or DBG
LIAB ApS <www.liab.dk>, Nov. 2006
root@liab# nanoctrl -l 5   < light LEDS D1 and D3 >
root@liab# nanoctrl -r
SW2:0 SW3:1
```

4.7 A demo program using the module nanomod

You may use nanoLIAB module "nanomod" from shell scripts, either by accessing it directly or by using the program nanoctrl. To use the module in a compiled C program, however, a series of standard C function calls is to be used. The relevant function calls in this demo program are "open()", "close()", "read()", and "write()".

The following program shows how to write a program making a running light on the four LEDs D1-D4 as long as none of the switches SW2-3 are depressed.

The first task in the program is to get a file descriptor for the device-special-file we want to communicate with. A file descriptor can be obtained using the "open()" function call. The corresponding function "close()" releases the file descriptor when it is no longer used.

An example of opening and closing a port to the nanoLIAB module is shown below. The actual communication with the module is done using the file descriptor and the two function calls "read()" and "write()". Please also consult the source codes located in the directory:

```
software/liabkernel/liab-modules/liabarmmod/userspace
```

on the accompanying CD-ROM.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define READBYTES 4

int main(void)
{
    /* This is the file descriptor */
    int fd;
    /* Buffer to hold characters to write and those read */
    char str[20];
    int i=1, cnt=0, sw=0;

    /* Try to open the device in read/write mode */
    fd = open("/dev/nanomod", O_RDWR);
    /* Check whether it opened correctly */
    if(fd < 0)
    {
      printf("Error opening module\n");
      exit(0);
    }

    while(1)
    {
        /* Read button states */
        do
        {
            cnt = read(fd, str, READBYTES);
            if(cnt > 0)
            {
                /* Parse the button states */
                if (strncmp(str, "00") == 0)
                    break;
            }
            usleep(100000);
        } while(1);

      /* Write the LED value to the module */
      sprintf(str, "^V%x", i);
      write(fd, str, strlen(str));
      i = (i == 8) ? 1 : i << 1;
      usleep(100000);
    }
    /* Close the device again */
    close(fd);
}
```

5. The Boot Loader

The boot loader represents the very first code executed after a power up or reset of the Atmel AT91RM9200 ARM microprocessor. The flow of the boot loader is shown in Fig. 5.1.

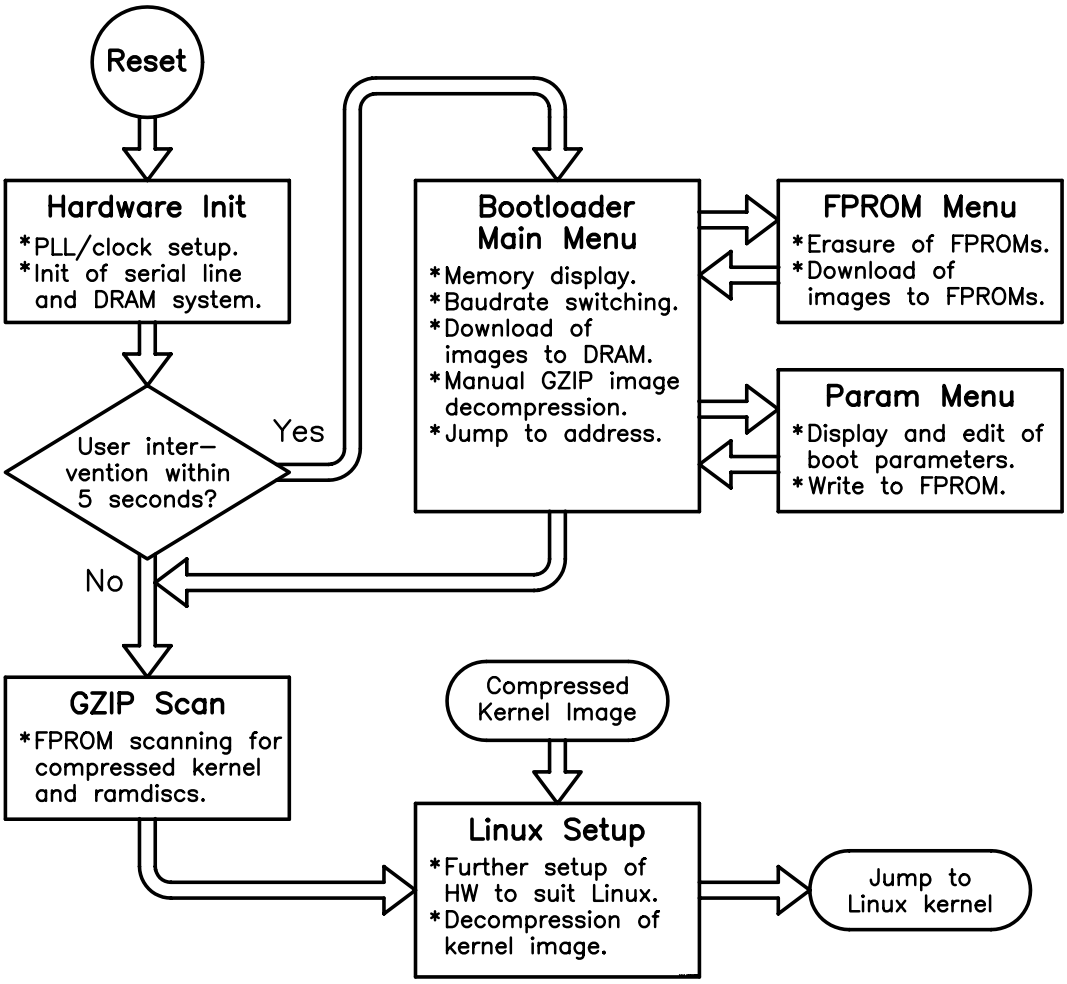
The firmware of the AT91RM9200 can use several sources for binary boot code: download using the debug serial port, fetch of code from the EEPROM or boot directly from the FLASH PROM.

When instructed to boot from the FPROM directly, the ARM processor starts executing instructions from address `0x0`. Now, the bootstrap written in assembler sets up debug serial port the DRAM system. Subsequently, the part of the FPROM that represents the boot loader is copied to DRAM and a execution is transferred hereto. Last, a stack segment is set up and a call is made to a "`main`". All further coding can be done in the C programming language, compiled using the `gcc` cross compiler. As discussed in section 3.3, the boot loader next prints a banner before it waits for five seconds, looking for three dots to be received over the debug port.

5.1 Three Dots Received ...

If in fact the boot loader receives the three dots within the five second period, the boot loader enters a menu system. The boot loader gives you a variety of options for display of memory, baud rate switch, code download, manual decompression of gzipped images and unconditional jumps to a prescribed location of memory.

In addition, you may enter two sub-menus, one for operations on the FLASH PROM and one for editing of the boot parameters. In the FLASH PROM menu you may read, erase and write to the PROM. In addition, you may download binary images which are programmed into the FLASH on the fly. The sub-menu for the parameters you may view, delete and enter new parameter strings which are given to the Linux kernel at boot time. To ease the entering of network parameters an interactive questionnaire is implemented in this sub-menu, see section 3.4.



Figur 5.1: *The boot loader for the nanoLIAB board.*

5.2 No Dots Received ...

If the five seconds elapse without the reception of three dots, the boot loader will try to boot a Linux system if one is found in the FLASH PROM. Both the Linux kernel and its accompanying ramdisk are expected to be in compressed state. The boot loader will now search the FLASH PROM for compressed images. In the event that a Linux kernel image is found, it will be decompressed into DRAM starting at address `0x20008000`. Further setup of the hardware is done (initialization of interrupt controllers, copy kernel parameters to DRAM at `0x20000100`, ...) before a jump to address `0x20008000` is performed. At this stage the Linux kernel takes over. The boot loader code is not used before a hard reset condition again is enforced on the Atmel AT91RM9200 ARM microprocessor.

5.3 Download of Binary Images

To download binary images over the serial port, you face the problem that most serial drives are unwilling to accept and transmit all the 256 possible ASCII characters: `0x00` to `0xff`. A simple way to solve the problem is to chop the stream of bits into chunks of six bits. With a proper offset, these chunks can now be send using the alphanumeric part of the ASCII codes. Other characters can be used to signal start-of-line, end-of-line, etc. For this purpose, a utility called `uuencode` is readily at hand in typical Unix or Linux systems. Thus, the "`l`"-commands expect the peer to transmit data produced by the `uuencode` program. Traditionally, the first line of a uuencoded stream specifies the filemode and filename like this:

```
begin 644 vmlinux.gz
M^F8/‘>!F@^‘!=‘7IZP‘‘‘.....
.........................
‘
end
```

However, no filename is needed in this context and the string representing it is instead used to specify the load address and a POSIX.1 CRC checksum in the format `<addr>-<crc>`:

```
begin 644 f0000-1150042577
```

where `f0000` is the load address in hex and `1150042577` is the CRC checksum in decimal. During load, the checksum of the binary data will be calculated and compared to the original checksum stated in the first line of the stream. You can generate a file to be downloaded using a script like this (named e.g. `mkuu`):

```
#! /bin/bash
FILE=$1
LOADADDR=$2
CKSUM=$LOADADDR-‘cksum $FILE | cut -f1 "-d "‘
uuencode $CKSUM < $FILE
```

which is called like this:

```
user@host$ mkuu <filename> <loadaddr>  >  <uufile>
```

6. MTD and JFFS2

Unlike a mainstream PC the LIAB board stores not only its bootloader, but also the Linux kernel and initial ramdisk image in the non-volatile part of memory: the FLASH PROM (FPROM for short). Normally, the FPROM is only accessed during boot-up since all relevant data are copied from FPROM to RAM during boot. When done, all further accesses are done to RAM. The downside of keeping all data in RAM is that the content of RAM disappears if the power is lost or the LIAB board is reset. It is therefore desirable to use part of the FPROM as a hard-disk like device: a FLASH disk.

Linux-kernels nowaday have support for FLASH disks using the driver system called "Memory Technology Devices" or MTD for short. By including MTD in the Linux kernel, access to the FPROMs becomes possible trough a set of char and block device special files. One can now create a filesystem on one of these block devices and next mount using a suitable mount-point. However, using a traditional Linux filesystem like `ext2` has at least two drawbacks: The first is related to power fails whereas the second is about wear for the FPROM chips.

During power fail, traditional file systems get corrupted and must be checked and repaired when the computer comes up again. Sometimes the repair even fails! Embedded systems are normally required to be tolerant to power fails, meaning that the system must be able to boot without problems when power is restored. Thus the filesystem used must be tolerant to interruptions. By keeping a journal of all transactions performed to the file system, its integrity can always be restored when the computer boots again.

Traditional filesystems have no strategy for the use of disk space. Thus, some sectors of the disk may be rewritten again and again, whereas other are left untouched for long periods. This is not a problem when dealing with hard disks, but each sector of a FPROM can only stand a limited number erase/rewrite cycles. To avoid erasing the same sector over and over again, a wear leveling algorithm must be used. The filesystem `JFFS2` (Journalling Flash File System 2) is a filesystem suited for FPROM since it utilizes both the above mentioned journalling principle as well as wear leveling.

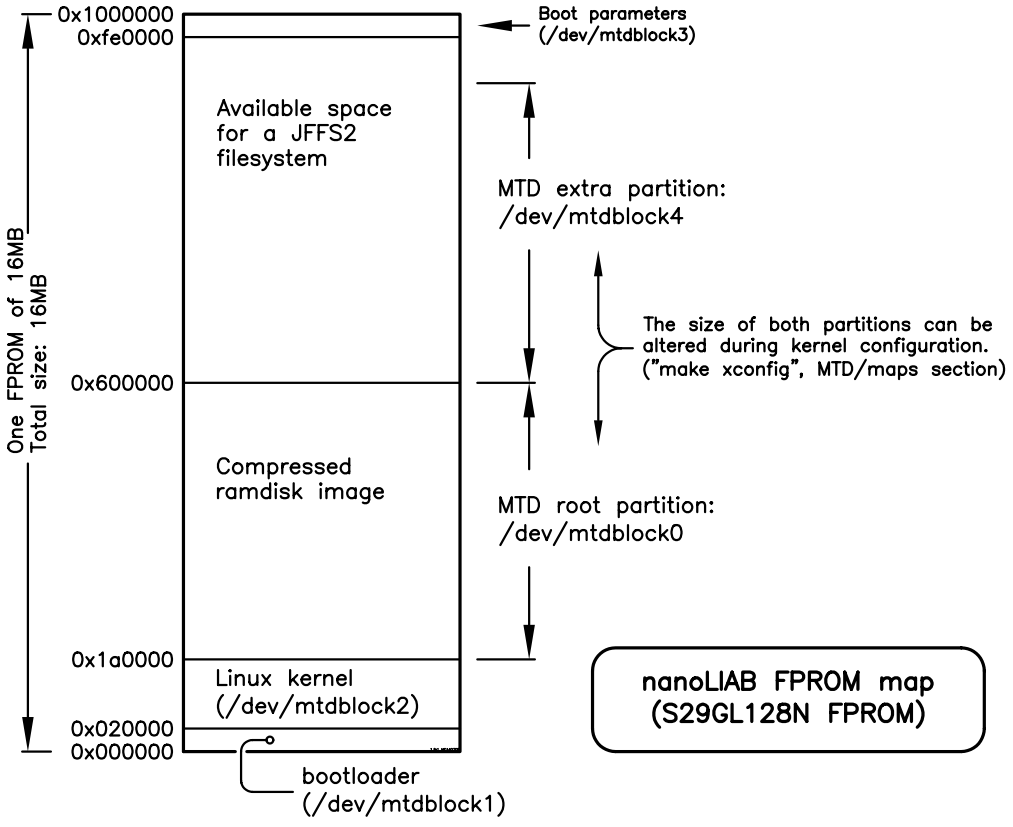
By employing MTD and the JFFS2 filesystem on the nanoLIAB, one may get

between two to ten megabytes of FLASH disk, depending of the hardware configuration of the nanoLIAB microprocessor board. This area is suitable for the storage of application-specific programs and data. The rest of this section deals with these systems on the nanoLIAB board.

6.1 Memory Technology Devices, MTD

Note: before carrying out the procedures listed below, you should check if your nanoLIAB has a suitable configuration for MTD and JFFS2.

The Memory Technology Devices system (MTD) gives access to FPROM, RAM and other types of memory through a set of block devices drivers under Linux. The MTD-system is part of Linux-kernel, and is now enabled by default in the standard LIAB Linux-distribution. In Fig. 6.1 you may study the default FPROM memory map on the nanoLIAB.



Figur 6.1: *Layout of FPROM memory on a standard nanoLIAB.*

The MTD partitions can be accessed from a user-space Linux program using the following device special files (the numbers in parenthesis denotes the major- and minor-number):

```
Partition:   char:               block:
Root         /dev/mtd0 (90/0)   /dev/mtdblock0 (31/0)
Boot         /dev/mtd1 (90/1)   /dev/mtdblock1 (31/1)
Kernel       /dev/mtd2 (90/2)   /dev/mtdblock2 (31/2)
Param        /dev/mtd3 (90/3)   /dev/mtdblock3 (31/3)
JFFS2        /dev/mtd4 (90/4)   /dev/mtdblock4 (31/4)
```

6.2 Journalling FLASH File System 2, JFFS2

The MTD system gives the ability to create and mount file systems on top of MTD block device special files. However, one must face a number of conditions when using FPROMs as the media for a file system:

1. Reading from a FPROM is nearly as fast as reading from static or dynamic RAM. However, writing to a FPROM is slow and in order to alter the content of a FPROM sector (typically 64 Kilobyte) one must first erase and then rewrite it.
2. Each sector of a FPROM is only able to tolerate a finite number of erase cycles, in the order of $10^5 - 10^6$ erasures.
3. FPROM memory systems are often used in embedded systems, where it is a requirement that the file system does not get corrupt when the processor is stopped abruptly by a power fail condition.

Problems regarding the slow writing speed can be solved using a proper buffering system and by having a pool of preerased sectors at hand. The overall lifetime of the FPROMS can be extended using wear leveling algorithms where a randomly chosen, preerased sector is used when data areas are to be updated. Last, by employing a transaction journal, the integrity of the file system can be reestablished when the Linux system is waked up again.

The most extensive FLASH file system under Linux is named "Journalling FLASH File System 2", JFFS2 for short, and is developed by RedHat. Support for this file system is selected during kernel configuration under "`File Systems`".

Bibliography

[1] D. P. Bovet and M. Cesati, *"Understanding the Linux Kernel"*, O'Reilly & Associates, Inc., Sebastopol, CA, first edition, 2001, ISBN 0-596-00002-2. 8

[2] M. K. Dalheimer and L. Kaufman, *"Running Linux"*, O'Reilly & Associates, Inc., Sebastopol, CA, fifth edition, 2006, ISBN 0-596-00760-4. 8

[3] R. Bentson, *"Inside Linux"*, SSC, Inc., Seattle, WA, 1998, ISBN 0-916151-89-1. 8

[4] A. Rubini, *"Linux Device Drivers"*, O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 2001, ISBN 0-596-00008-1. 8

[5] A. Rubini J. Corbet and G. Kroah-Hartman, *"Linux Device Drivers"*, O'Reilly & Associates, Inc., Sebastopol, CA, third edition, 2005, ISBN 0-596-00590-3. 8

[6] M. J. Bach, *"The Design of the UNIX Operating System"*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986, ISBN 0-13-201757-1. 8

[7] M. J. Rochkind, *"Advanced UNIX Programming"*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985, ISBN 0-13-011800-1. 8

[8] P. K. Andleigh, *"UNIX System Architecture"*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990, ISBN 0-13-949843-5. 8

[9] W. R. Stevens, *"UNIX Network Programming"*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990, ISBN 0-13-949876-1. 8

Links

Below, a number of relevant hyperlinks for ARM development are listed.

ARM General

The AT91 ARM resource pages are maintained by Atmel. It contains information on both Atmel development kits for their ARM series, a support forum, and links to third party development tools.

http://www.at91.com

ARM Linux

ARM Linux is a port of the Linux Kernel to ARM processor based machines, lead mainly by Russell King, with contributions from many others. ARM Linux is under almost constant development by various people and organizations around the world.

http://www.arm.linux.org.uk

In the ARM Linux developer section the newest ARM kernel developments are available, plus additional useful information for ARM development on Linux.

http://www.arm.linux.org.uk/developer/

Debian Linux

The nanoLIAB distribution is based on Debian Linux.

http://www.debian.org

The more than 15000 packages for the Debian Linux distribution are distributed from several mirrors, most of which are listed on the Debian Package pages. Most of these packages already exist in readily downloadable versions compiled for ARM.

http://packages.debian.org

A. Using cu as terminal emulator

The nanoLIAB was specifically designed for the Linux operating system and a natural choice for a development platform and host computer would be an IBM-compatible Personal Computer (PC) running Linux. A number of terminal emulators are readily available in the various Linux distributions or can be downloaded over the Internet.

One of the oldest and simplest emulators is the cu-program, which stands for "Connect Unix". You don't get any fancy graphical user interface, you just get connected!

In the following, we assume that the nanoLIAB is connected to the COM1-port on the host PC and that this port can be accessed trough the device file named `/dev/ttyS0` (this is at least true for Redhat 7.x and 8.x distributions).

To start cu, log in as root and type the command:
"`cu -l /dev/ttyS0 -s 115200`". To exit cu again, type a tilde: "~", followed by a dot: "`.`" on a new line:

```
user@host$ cu -l /dev/ttyS0 -s 115200
Connected.
  < communication with the LIAB board >
~.
Disconnected.
user@host$
```

It may be inconvenient to have to log in as root when using cu. To access the COM1-port from any user account, you have to change the permissions on the device files that refers to the COM-ports: "`chmod 666 /dev/ttyS0`" will give permissions to everybody to use the port.

Downloading kernel- or disk-images to the LIAB board can also be done from within cu. If you want to download the image-file "v", located in the same directory as where cu was started, you first tells the remote system, in this case a LIAB, that a download is to be initiated. Next, by issuing the command ~>v the download is started. A typical example is given on the next page:

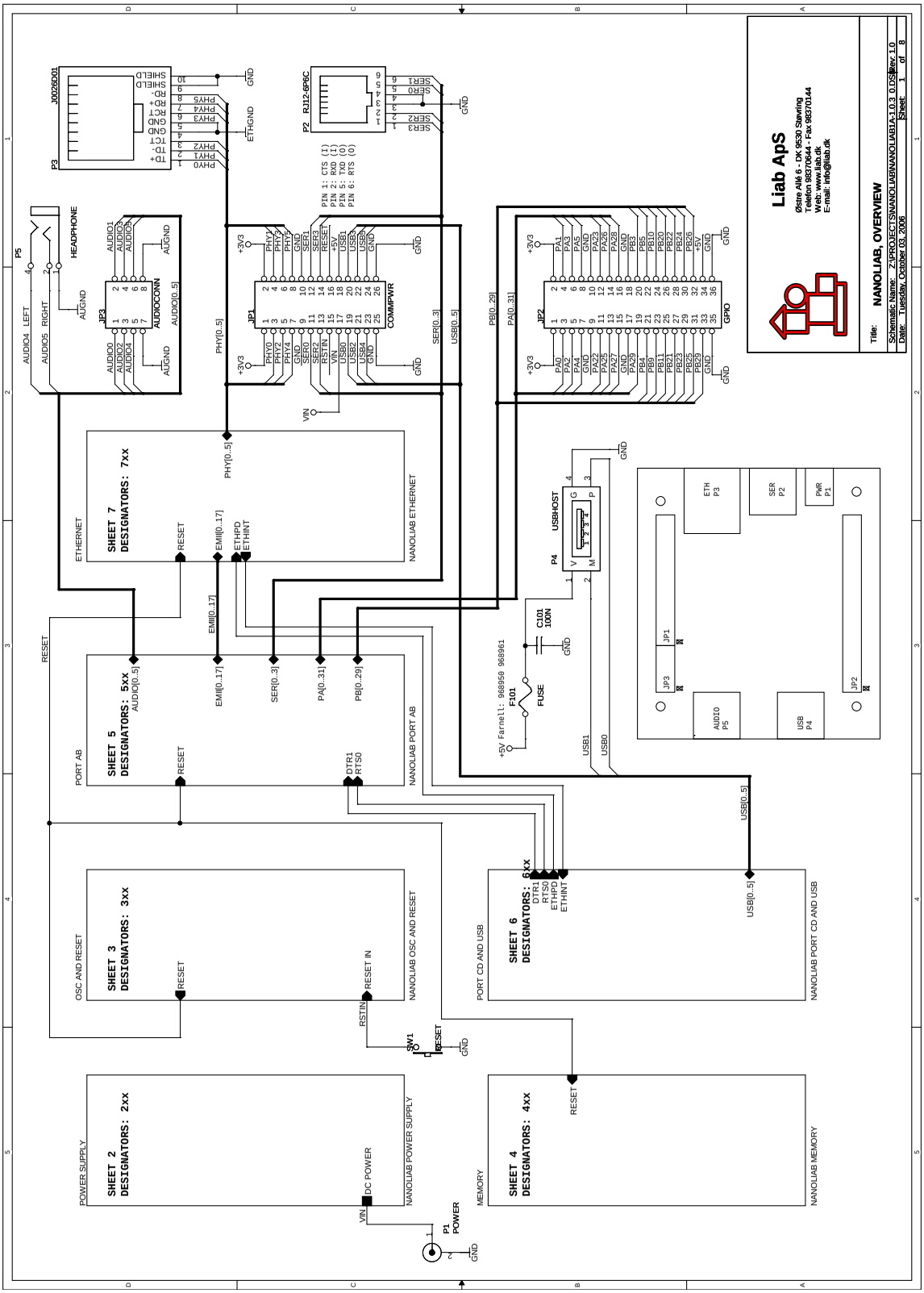
```
user@host$ cu -l /dev/ttyS0 -s 115200
Connected.
  < communication with the LIAB board >
29F800>l
~>v
1 2 3 4 5 6 7 8 9 10 11 ....   < indication of download >
....  ....  ....  .... 862 863 <  or some other number  >
[file transfer complete]
[connected]
~.
Disconnected.
```

B. Schematics and Layout

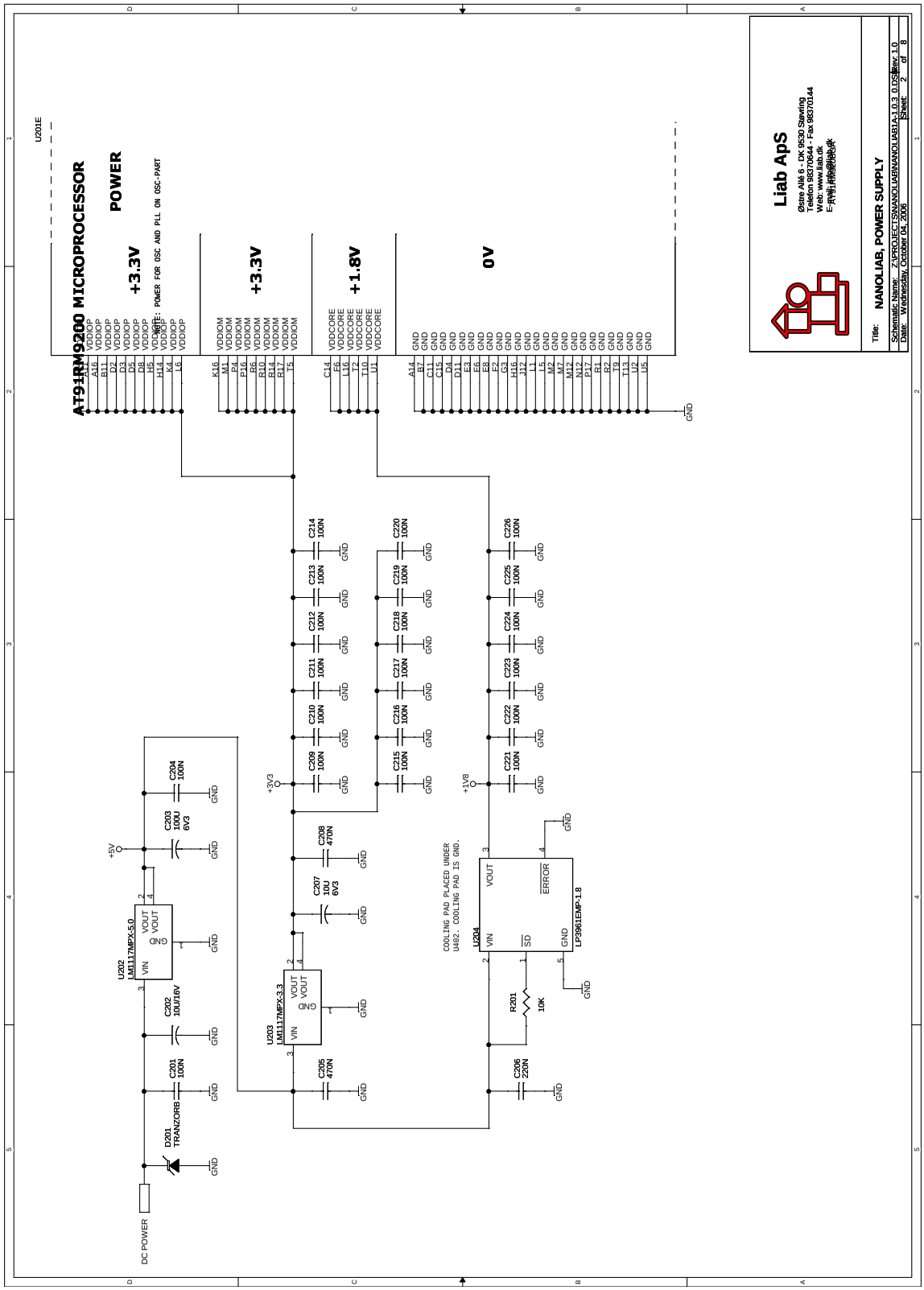
On the following pages the complete schematics and component layouts for the nanoLIAB microprocessr board.

Figure no.	Description:
B.1	nanoLIAB: Block diagram, connectors, pin-headers.
B.2	nanoLIAB: Power supply and decoupling.
B.3	nanoLIAB: Reset circuit and CPU oscillators.
B.4	nanoLIAB: AT91RM9200 CPU, FLASH and SDRAM memory.
B.5	nanoLIAB: CPU PIO port A and B, Serial port, RTC and audio system.
B.6	nanoLIAB: CPU PIO port C and D, USB system, LEDs, switches.
B.7	nanoLIAB: Ethernet PHY.
B.8	nanoLIAB: Component placement, top.
B.9	nanoLIAB: Component placement, bottom.

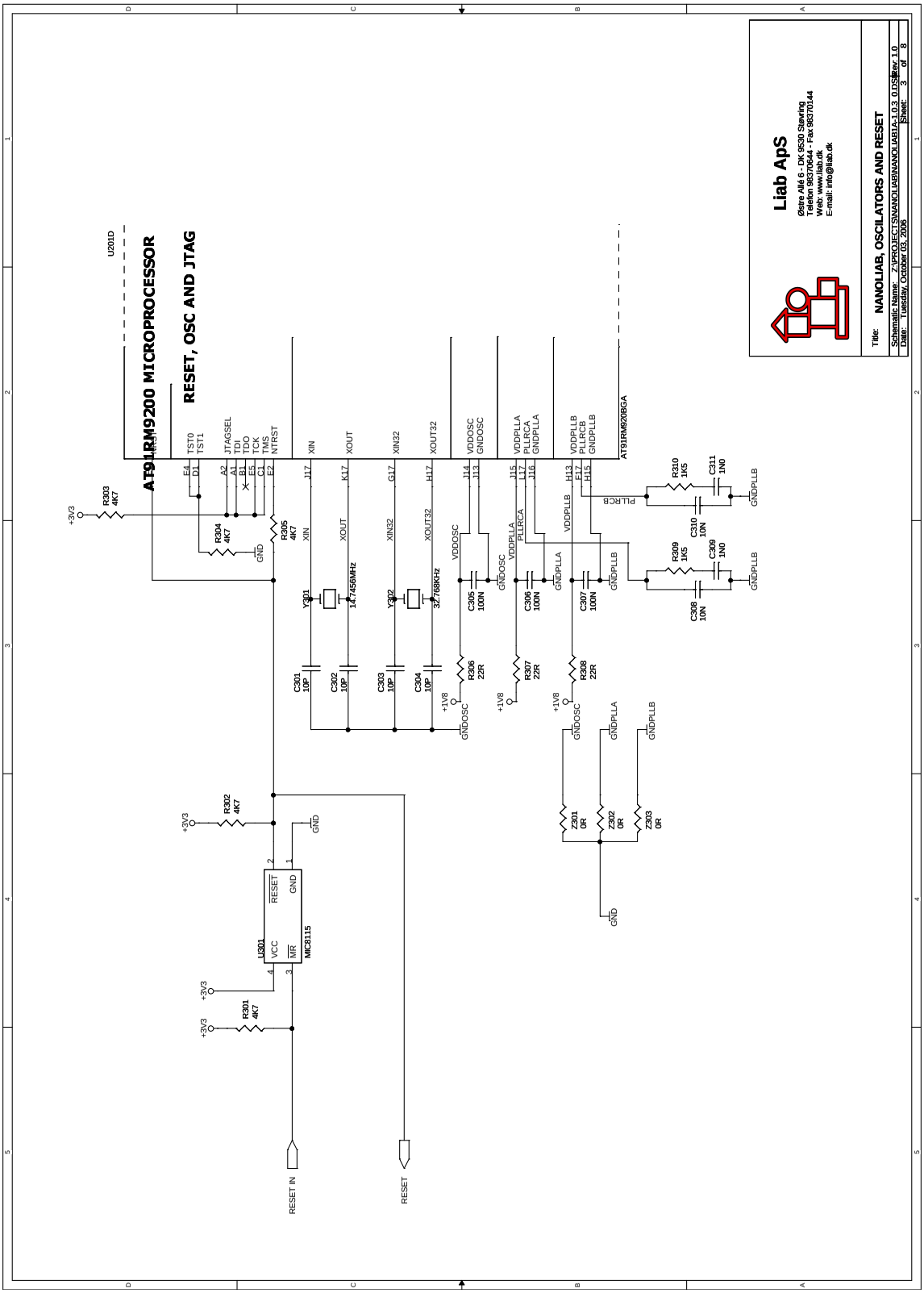
Tabel B.1: *Overview of schematics and component placements on the following pages.*



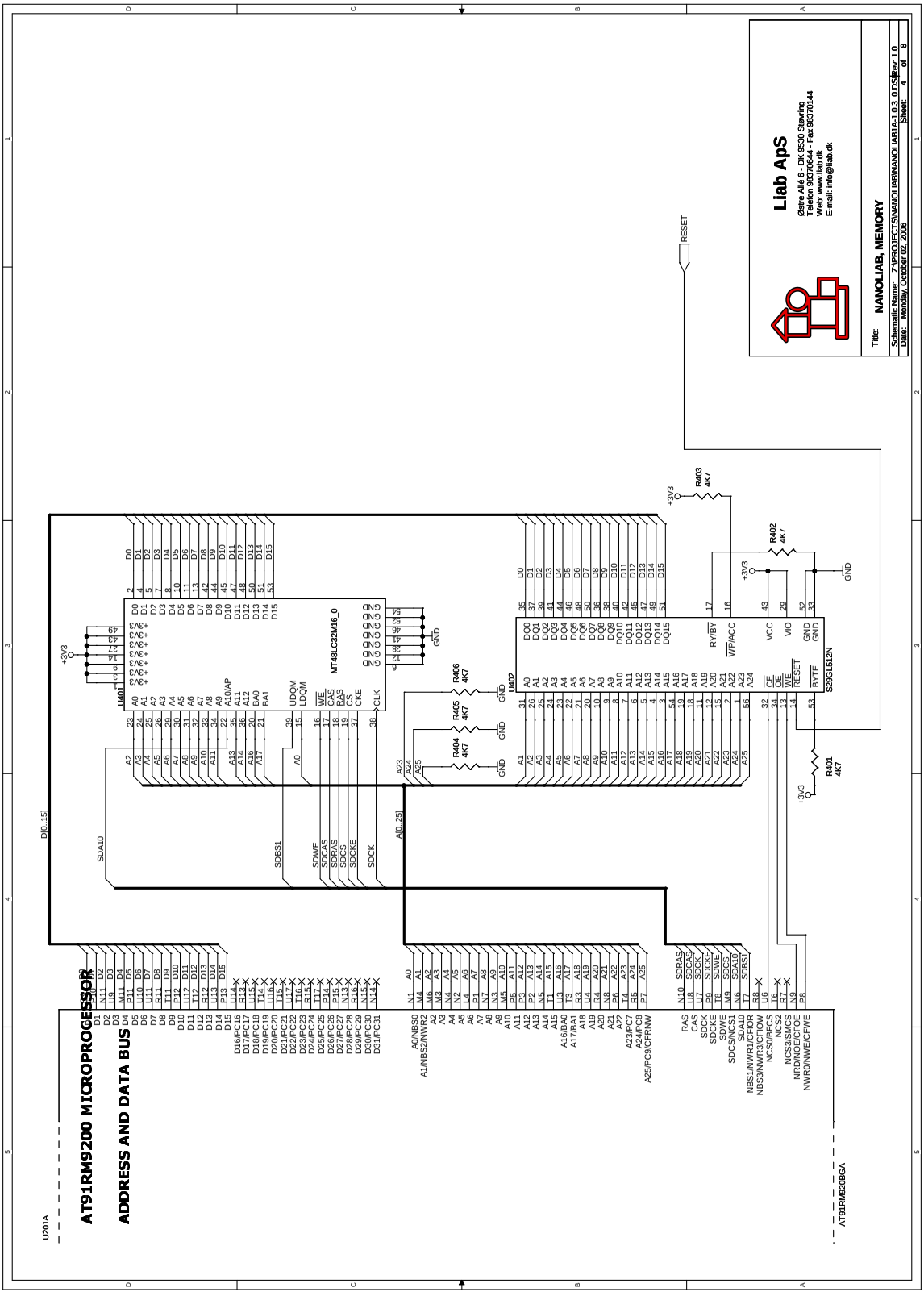
Figur B.1: *nanoLIAB: p. 1: Block diagram, connectors, pin-headers.*



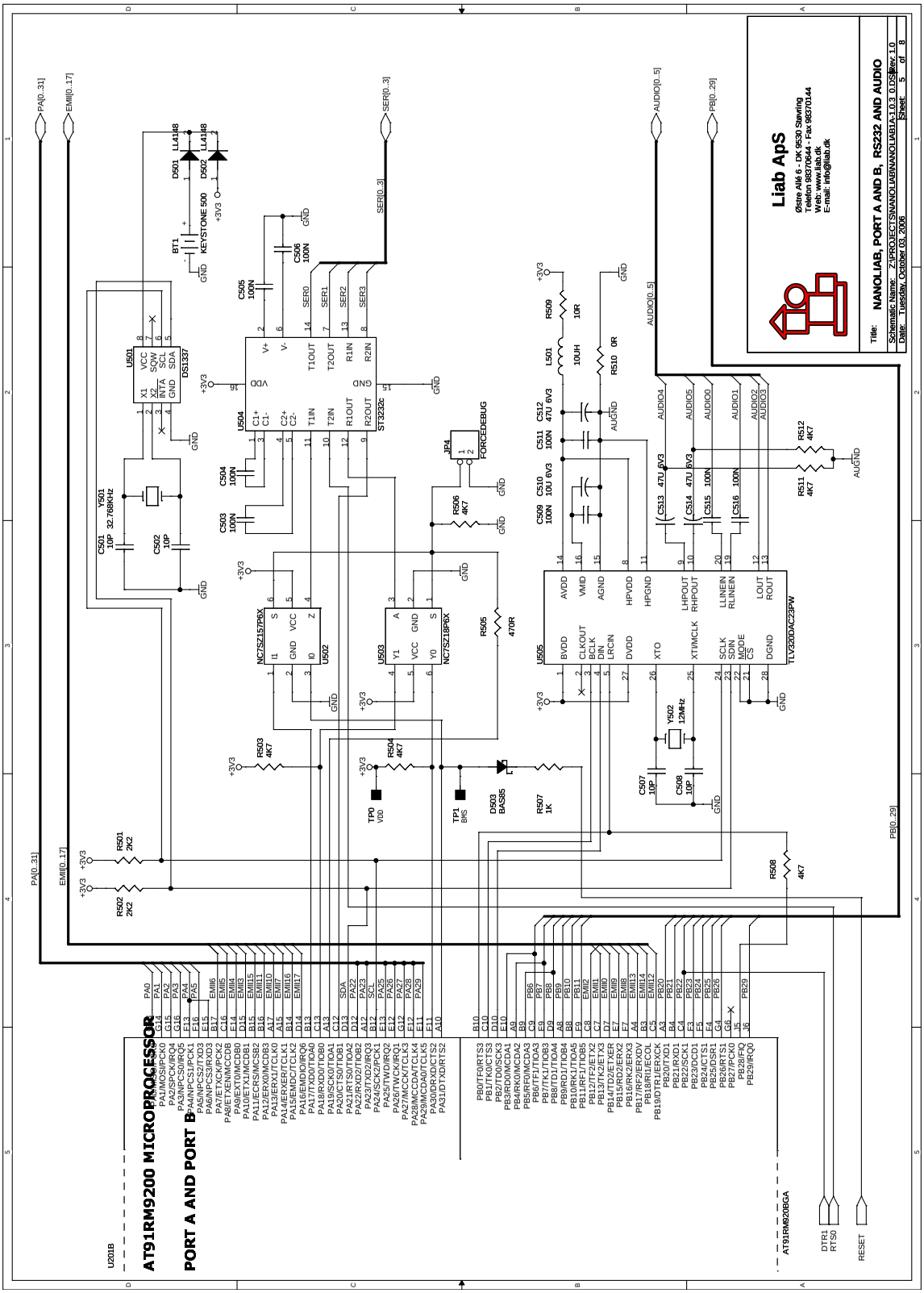
Figur B.2: *nanoLIAB: p. 2: Power supply and decoupling.*



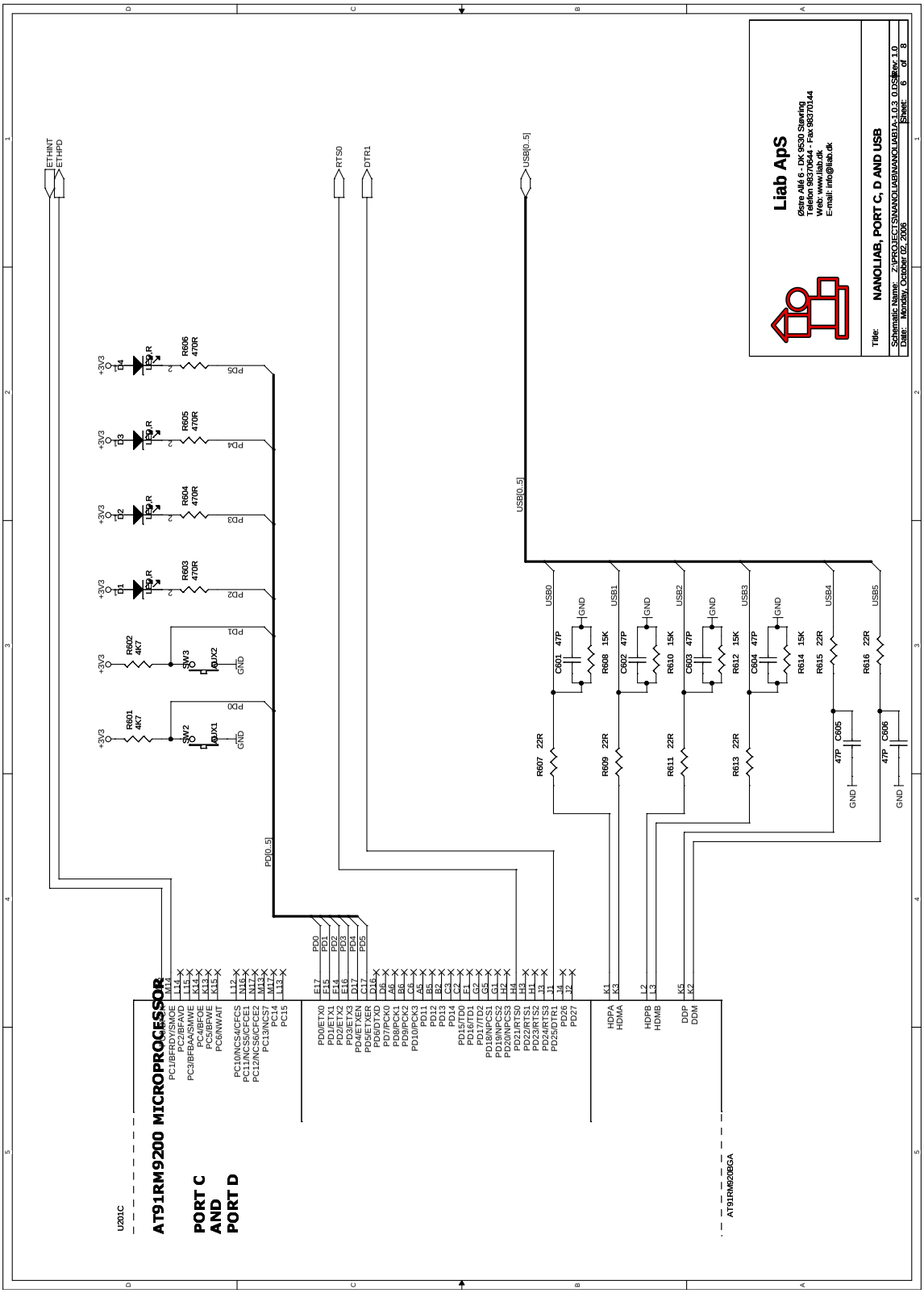
Figur B.3: *nanoLIAB: p. 3: Reset circuit and CPU oscillators.*



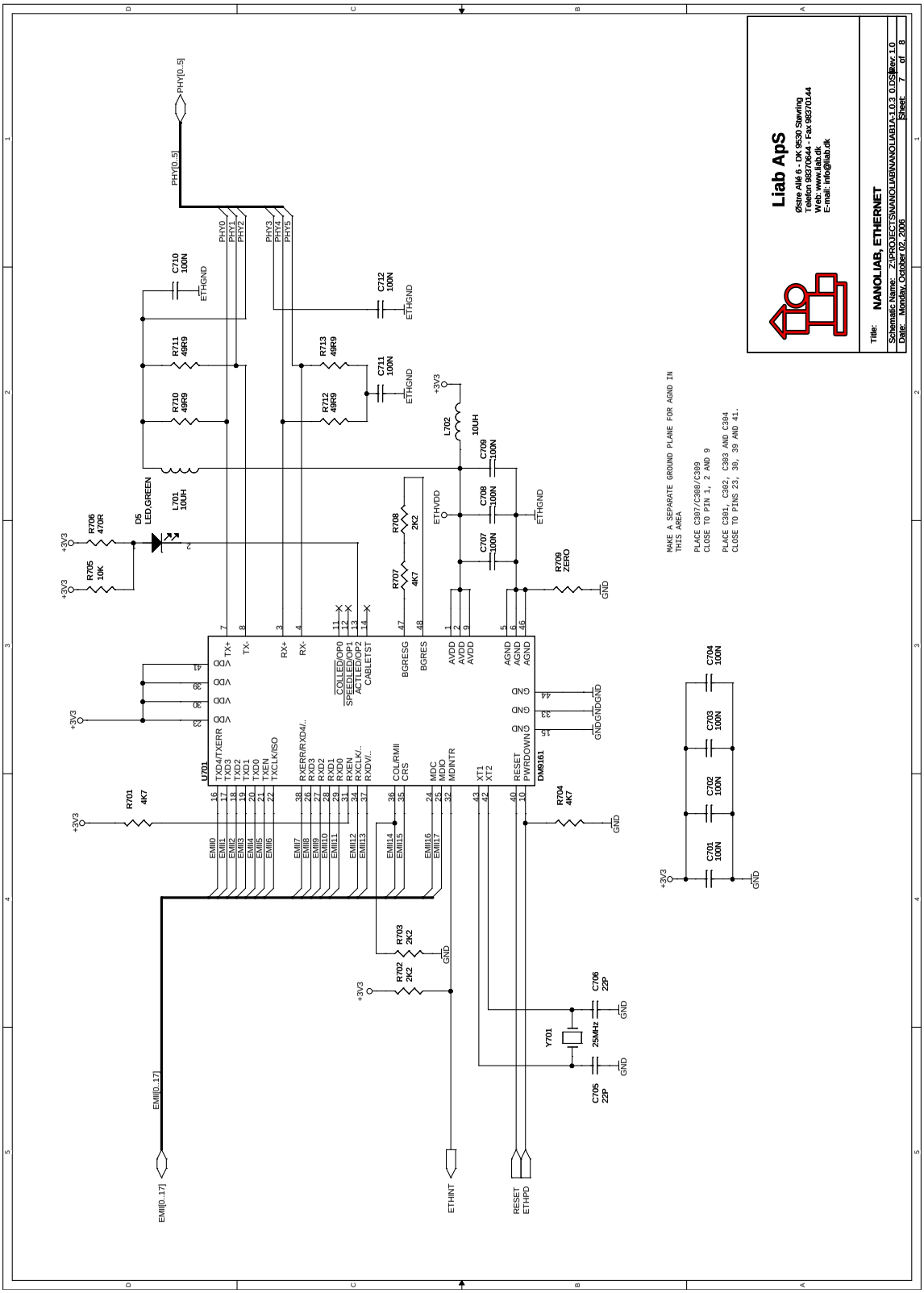
Figur B.4: *nanoLIAB: p. 4: AT91RM9200 CPU, FLASH and SDRAM memory.*



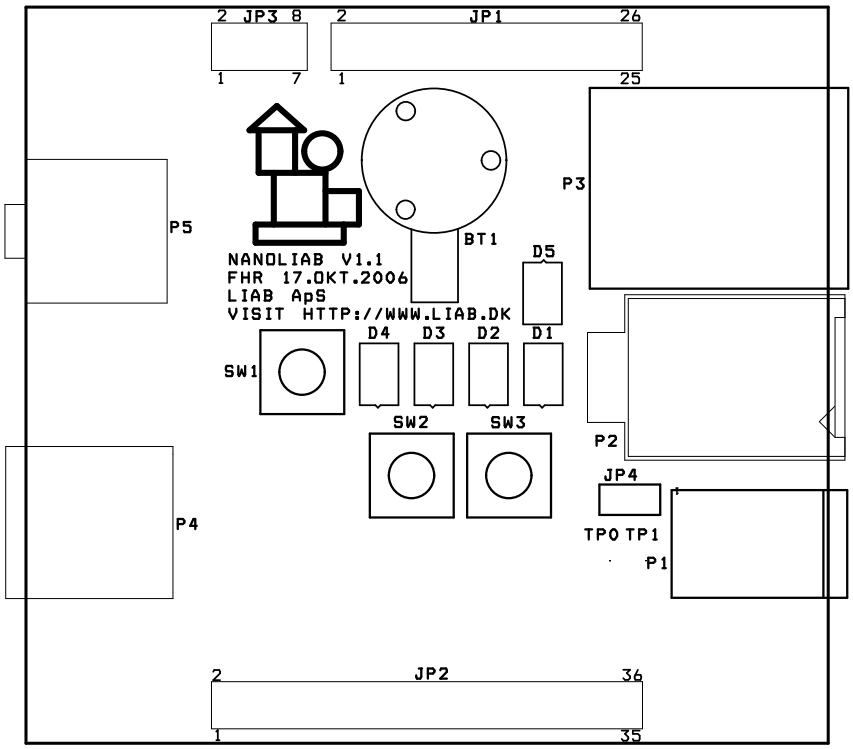
Figur B.5: *nanoLIAB: p. 5: CPU PIO port A and B, Serial port, RTC and audio.*



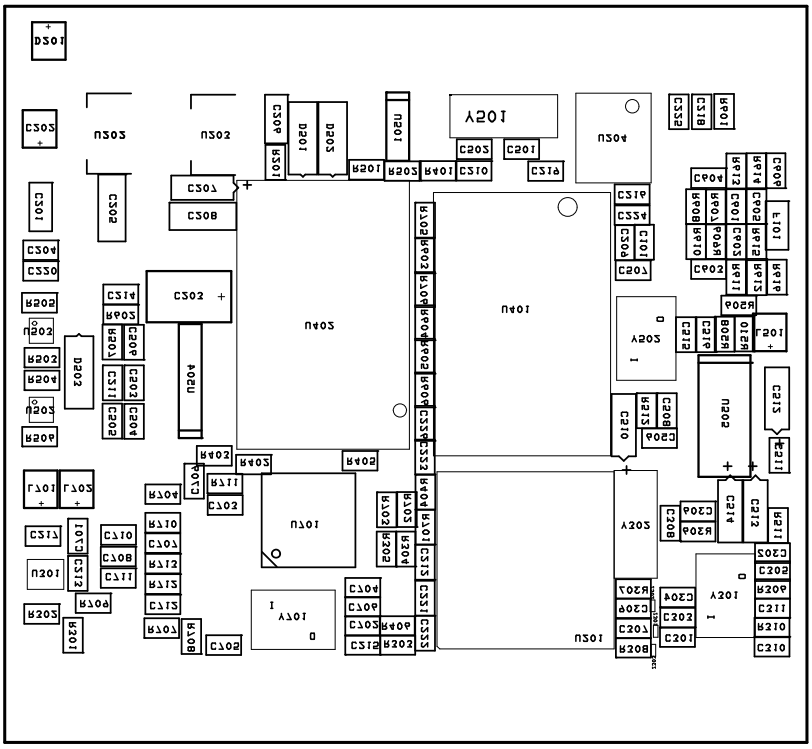
Figur B.6: *nanoLIAB: p. 6: CPU PIO port C and D, USB, LEDs, switches.*



Figur B.7: *nanoLIAB: p. 7: Ethernet PHY.*



Figur B.8: *nanoLIAB: Component placement page 1: Top.*



Figur B.9: *nanoLIAB: Component placement page 2: Bottom.*