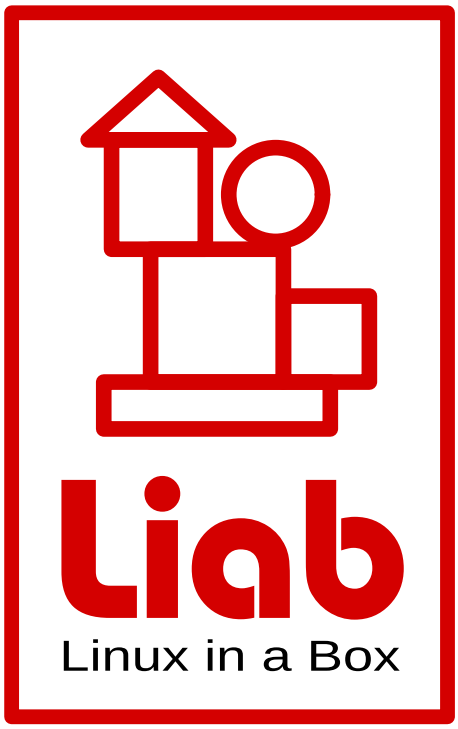


User's Manual for the

LIABARM9200

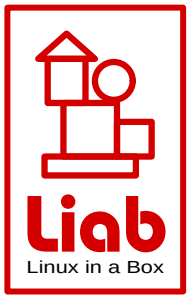
microprocessor board

Version 01.00, October 2005

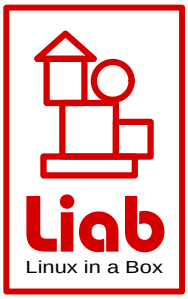
This document describes the use of the LIABARM9200 microprocessor board. An introduction to the LIAB (Linux In A Box) concept is given together with a description on how to get the microprocessor board up and running. Procedures for the development of software and hardware extensions are also given.

Please also consult the accompanying LIABARM-9200 CD-ROM for software distribution and schematics. Note that all parts of the LIABARM9200 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 two separate computer system elements which make up an evaluation platform for the LIABARM9200 system:

1. The LIABARM9200 microprocessor board itself. The board is 80×100 mm^2 and has a seven segment display and a hex switch in its center. Along the narrow sides, pin headers are located. No standard connectors for Ethernet, USB or serial ports are found on this board. A drawing of this board is shown in Fig. 2.2 on page 12. The board is preloaded with Linux when shipped from LIAB ApS.

2. The LIABARM9200 standard baseboard for evaluation purposes. The main object of this board is to route signals from the pin headers on the microprocessor board to standard connectors for Ethernet, USB and serial ports. Several costumers have designed similar baseboards which suit their particular needs.

For first time users: From LIAB ApS you may order either the microprocessor board alone or together with a baseboard. If you are a first time user of the LIABARM9200 system, it is most convenient if you have both boards, because you can communicate with the system using standard interfaces.

For the impatient user: If you are eager to experience the features of the Linux system on the LIABARM9200 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 LIABARM9200 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 2.4
...
```

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 LIABARM9200 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 LIABARM9200 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] , 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, [5] [6] [7] [8] 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 LIABARM9200 series microprocessor boards.

Based on the AT91RM9200, the LIABARM9200 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 two 50-pin interfacing connectors and one 20-pin media connector. 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 LIABARM9200 microprocessor board is identified as the top board.

Baseboards for the product microprocessor board are equipped with a standard set of interfacing options, to provide easy and effective communication with the surrounding world. On the LIABARM9200 standard baseboard the interfaces includes an 10/100 MBit Ethernet connector, two RS-232C serial ports, two USB host ports, one USB slave port and a CompactFlash® connector. The baseboard also provides a switched mode power supply, accepting between 7.5 and 24 VDC from any standard type power supply. The standard baseboard is identified as the bottom board on Fig. 1.1.

Currently, only the standard baseboard is available from LIAB. However, it is expected that more application specific baseboards will follow. These could include a multimedia baseboard, featuring both audio and graphics 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.

All references to the baseboard in the remainder of this document will be to the LIABARM9200 standard baseboard currently available from LIAB.

The LIAB board is presently distributed with a version 2.4.25 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 LIABARM9200 must be compiled for the ARM platform using a cross-compiler, e.g. the one included on the accompanying CD-ROM.

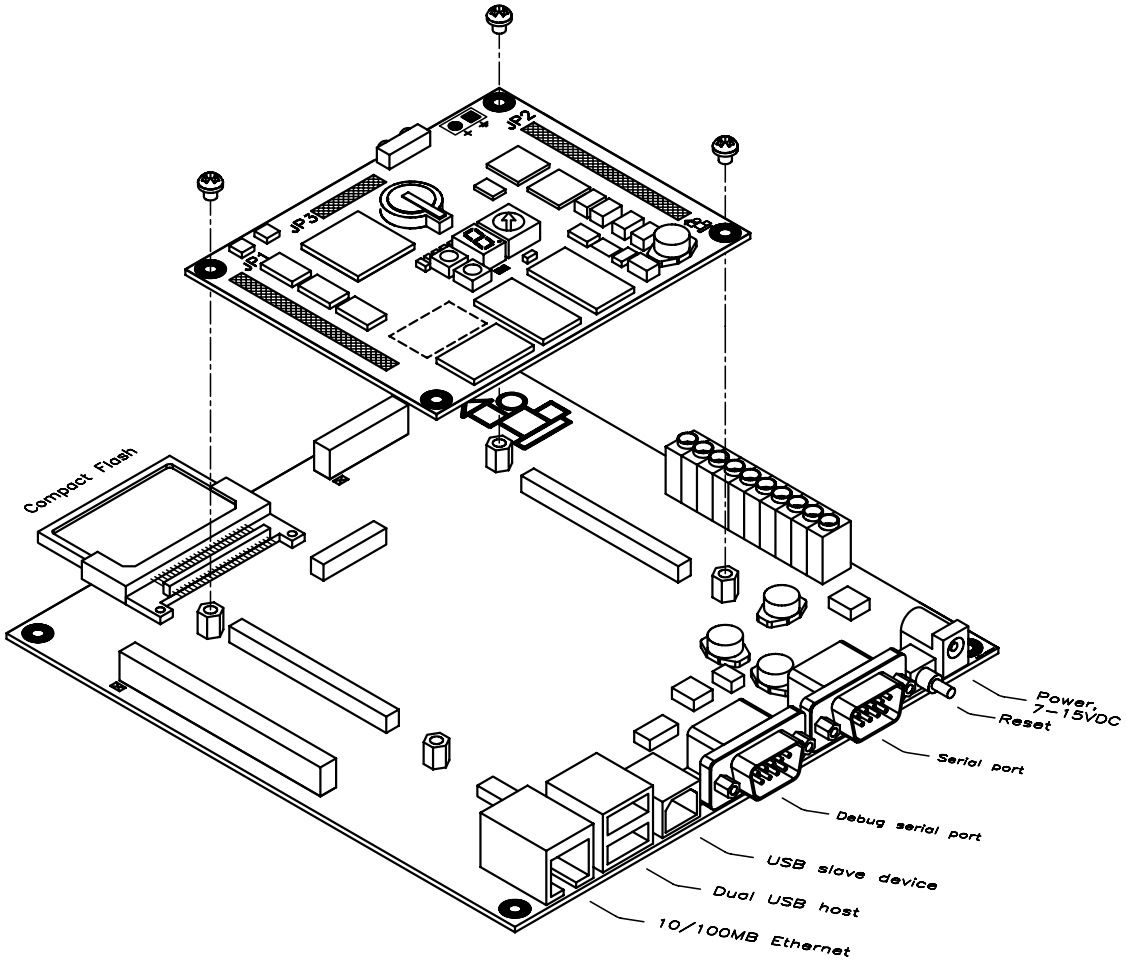


Figure 1.1: *The LIABARM9200 microprocessor board mounted on the standard baseboard. The microprocessor board is at the top.*

2. The LIABARM9200 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 LIABARM9200 evaluation system comprised of two modules, the microprocessor board and a baseboard as depicted in Fig. 1.1. The microprocessor board uses 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 (0.5-0.8 Watts). The on-board FLASH PROM memory provide storage for the bootloader, the Linux operating system and application software and data. The main purpose of the LIABARM9200 standard baseboard is to route the signals from the microprocessor board into standard connectors and interfaces.

First, the features of the LIABARM9200 microprocessor board are described together with a discussion of the signals in the three pin headers located on the bottom side of the board. Second, the LIABARM9200 standard baseboard is described

2.1 The LIABARM9200 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 switch mode power supply. To provide flexibility, all access to the features of the board is through three pin-headers located on the bottom side of the board. A block diagram of the LIABARM9200 microprocessor board is shown in Fig. 2.1.

The LIABARM9200 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 interface, and general purpose digital I/O ports. The AT91RM9200 features an address/data bus, which can be configured either as a general purpose ISA-like bus or as a PCMCIA-like bus. In addition several dedicated interfaces are present: Serial ports, Ethernet, USB host, USB slave, I^2C, SPI, ...

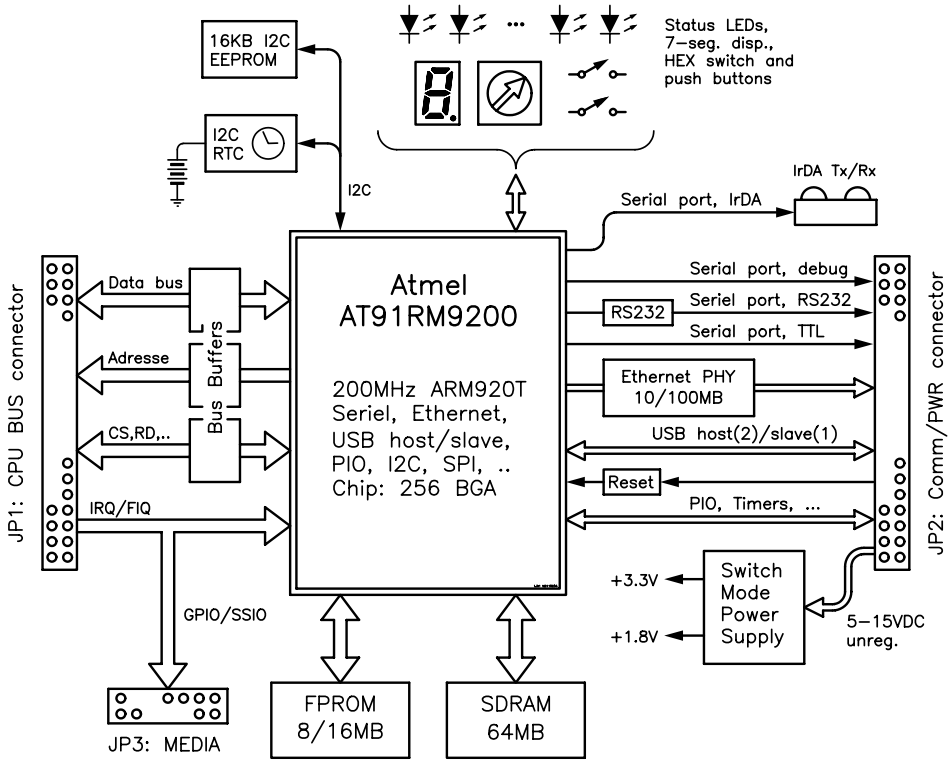


Figure 2.1: *Block diagram of the LIABARM9200 microprocessor board.*

- 8 MB (16 MB optional) non-volatile FLASH PROM memory for the boot loader and the Linux system.
- 64 MB synchronous DRAM functioning main work storage for the Linux system.
- Three asynchronous serial ports, one port acting as debug port and Linux console, whereas the two remaining are for general purpose. One of the general purpose ports employs a RS232C line driver, the two other ports are at TTL levels.
- An 10/100 Mbit Ethernet interface, complete with an Ethernet PHY. Thus, you just have to connect a Ethernet RJ45 socket with build-in transformers to the board to get it onto the net.
- A 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. The board also employs a USB slave port.
- IrDA interface, Real Time Clock (RTC) with battery backup and a I^2C EEPROM, suitable for configuration data.
- Human I/O interface having an 7-segment display, 10 LEDs of different colors, two push-buttons and a HEX switch.

- A switch mode power supply producing +3.3V for the on-board logic.
- Two 50 pins I/O connectors and one 20 pins media connector.

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 80×100 mm^2 (app. 3×4 sq. inches) and the weight is less than 100 grams. All components are placed on one side of the PCB, except for the two 50-pin I/O connectors and the 20-pin media connector.

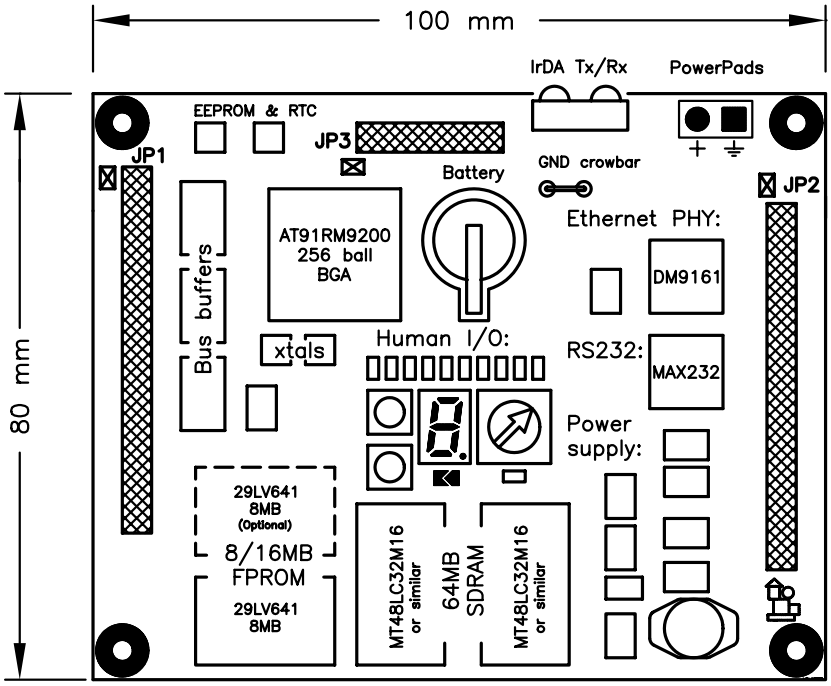


Figure 2.2: *Layout of the main components on a LIABARM9200 microprocessor board.*

The layout of connectors, jumpers and main components are shown in Fig. 2.2. The two 50-pin I/O-connectors and the 20-pin media connector are show as hatched rectangles on the figure. The purpose of the individual headers JP1-3 is described in Table 2.1 on page 13.

The microprocessor board must be powered by a DC supply having a voltage between 4.5 and 15 volts (note that this range is different from the voltage range for the standard baseboard). The board consumes about 0.8 watts and it has been tested to work in temperatures ranging from -10^{o}C to +60^{o}C.

2.2.1 LEDs and Switches

The LIABARM9200 microprocessor board is equipped with 10 LEDs (D1-10), two push buttons (SW1-2), a HEX switch (SW3), and a seven segment numerical display (DP1). All of these are fully programmable, and can be used as an operator interface for debugging and configuration purposes. In section 4.6 a Linux kernel module for the control of these elements are presented, together with some programming examples.

2.2.2 Pin Headers JP1 to JP3

The three pin headers present on the bottom side of the LIABARM9200 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	50	CPU BUS connector: • CPU data bus: 16 bits, `D0-D15` • CPU address bus: 14 bits: `A0-A10, A22, A23, A25` • CPU controls: `CS/RD/WR, IRQ/FIQ` • Power out: `+3.3V` All bus and handshake signals are buffered.
JP2	50	Communications and power connector: • 16 bit Programmed Input-Output (PIO) • Ethernet 10/100 MBit • 1 debug serial port at TTL level (`TX` and `RX` only) • 1 RS232C serial port (`TX, RX, CTS, RTS`) • 1 TTL serial port (`TX, RX, CTS, RTS, DTR, DSR, DCD, RI`) • 2 USB host, 1 USB slave • Reset input, timer functions • Power in: `4.5-15 V`
JP3	20	Media connector: • Programmable Input/Outputs (PIO) and Synchronous Serial Input/Outputs (SSIO) for media interfacing.

Table 2.1: *Pin-headers on the LIABARM9200 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.

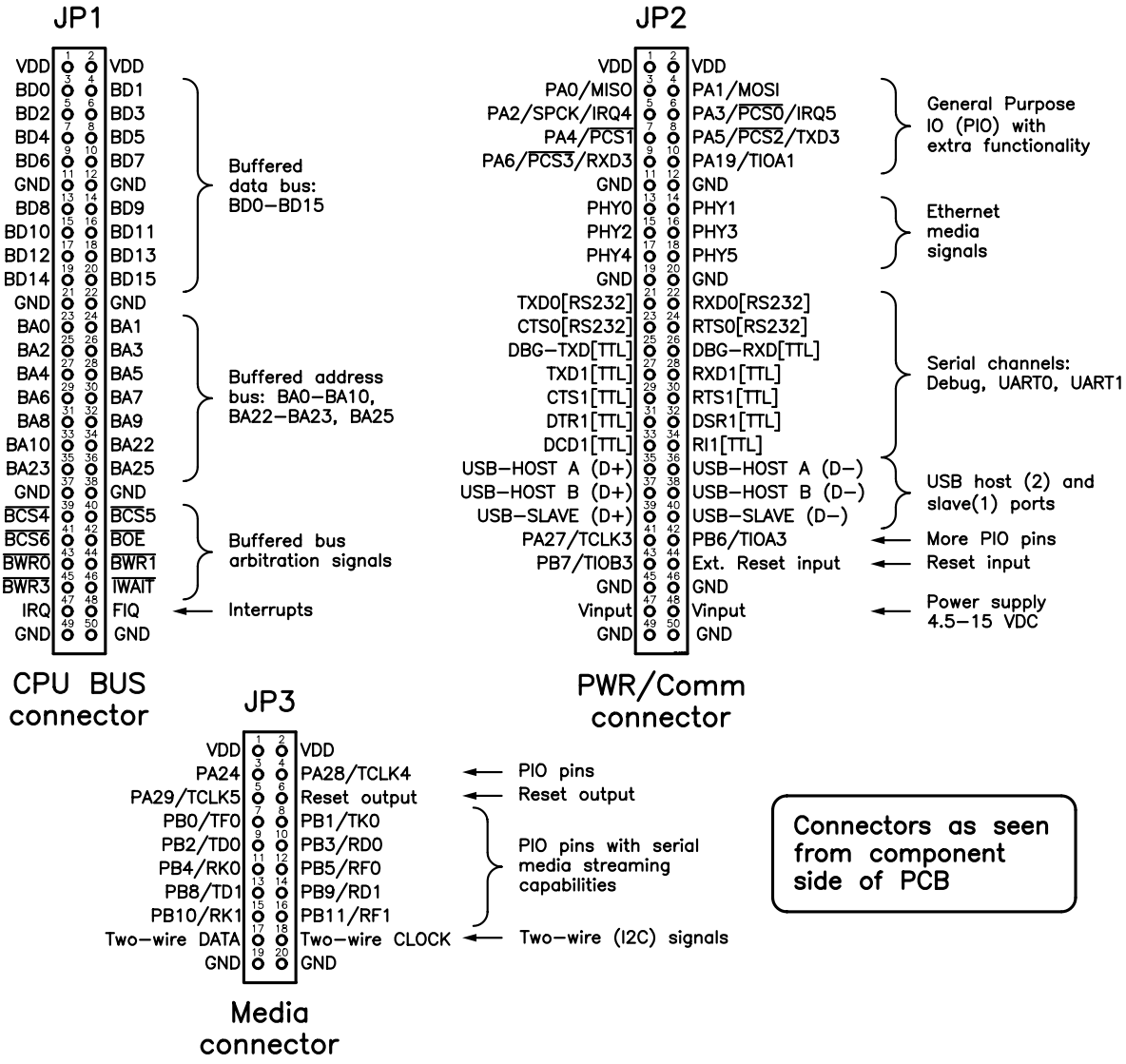


Figure 2.3: *Signals of the three connectors, JP1-3.*

2.3 The LIABARM9200 Standard Baseboard

The LIABARM9200 standard baseboard contains both a power-supply and physical interface hardware, such as a CompactFlash® socket, RJ45-connector, and USB host and and slave device connectors.

A block diagram of the standard LIABARM9200 baseboard is shown in Fig. 2.4 it has the following features:

- An on-board 10W 5V switch-mode power supply. Thus, all you need to run the system is an unregulated 7.5V to 24V DC supply. Standard DC power adapters of the wall-plug-in type are feasible.
- Two 50-pin and one 20-pin connectors which connects the LIABARM9200 microprocessor board peripherals to the appropriate connectors.

- Standard connectors including an RJ-45 Ethernet connector, two USB host type B connectors, one USB device type A connector, a CompactFlash® card socket.
- Accessible duplicates of the two pin headers JP1 and JP3. These pin headers are employs a 2.57mm (0.1 inch) module grid. You should be aware that neither pin headers are compatible with those of earlier LIAB boards.
- PCB outline of 160×160 mm^2 (approximately 6.3×6.3 square inches).

2.3.1 Board Layout and External Connections

The electronics for the LIABARM9200 standard baseboard is assembled on a two layer Printed Circuit Board (PCB). All the components are mounted on the top side of the PCB. The layout of connectors, jumpers and main components is shown in Fig. 2.5.

Power and external stimulus can be applied to the LIABARM9200 microprocessor board through the connectors and the reset button located on the LIAB-

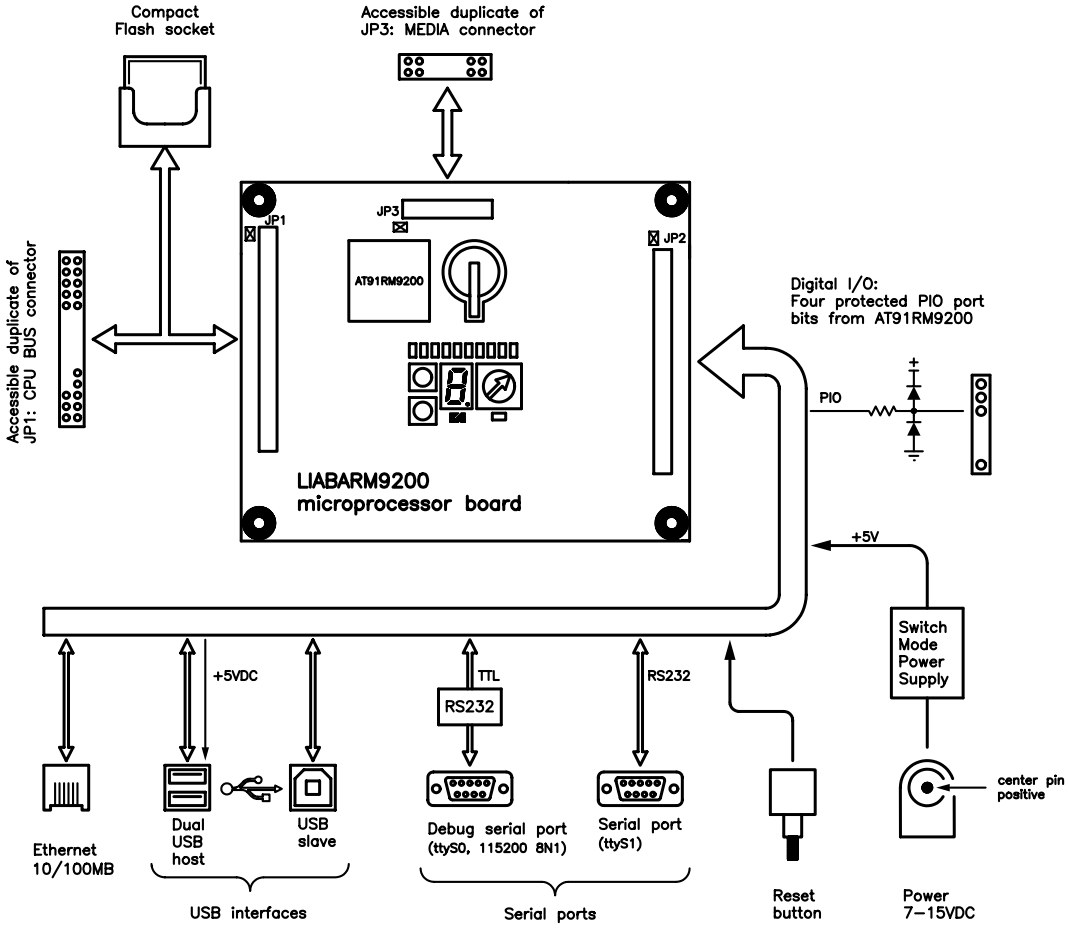


Figure 2.4: *A block diagram of the LIABARM9200 standard base board*

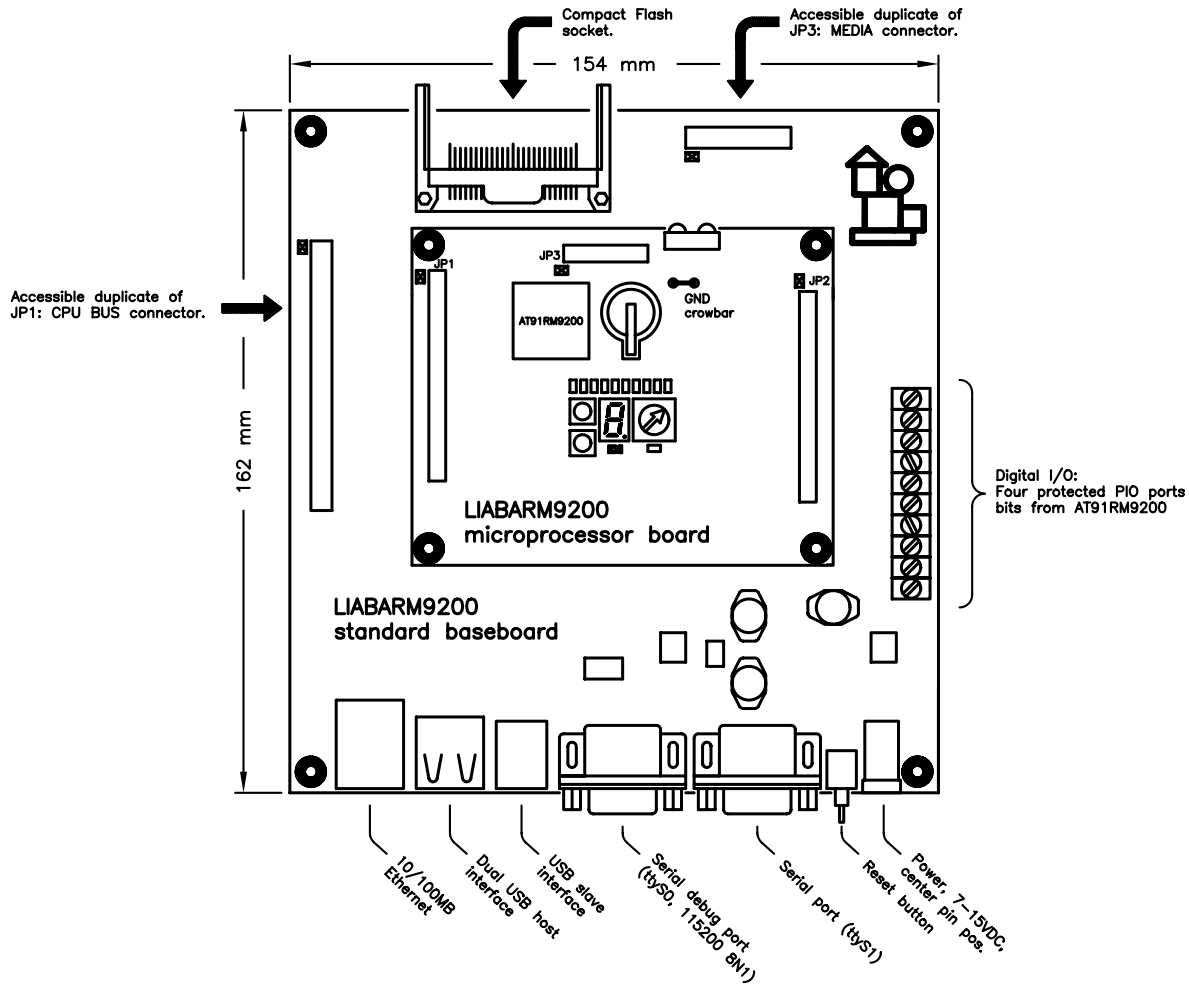


Figure 2.5: *Layout of the main components on a LIABARM9200 standard baseboard.*

ARM9200 standard baseboard. The connectors and switches on the standard baseboard are listed in Table 2.2.

In should be noted that a LIABARM9200 baseboard can be a relatively simple circuit, which means that persons with access to photo PCB equipment should be able to produce her own baseboard.

Item	Type	Purpose
J301	mini-jack	Power connection: The standard baseboard must be powered by unregulated DC supply between 7.5 and 24 volts. Center pin in the jack is positive.
SW2	push button	Reset button: Depressing this button resets the microprocessor and subsequently the bootloader will be entered. You can enter the bootloader menu by sending three dots ("...") over the debug serial port at 115200 baud within the first five second after reset. Alternatively, the bootloader will try to boot the Linux system.
P201	9 pins DB-9, male	Debug serial port: RS232 serial line: All communication with the bootloader is done trough the debug port. See text for the reset switch SW1 above. Linux-device: `/dev/ttyS0`.
P202	9 pins DB-9, male	UART0 port: RS232 serial line: A general purpose RS232 line. Linux-device: `/dev/ttyS1`.
U1	8 pins RJ45, female	Ethernet: 10/100Mbit Twisted-Pair (TP): High speed connections to local network or the Internet through a suitable switch or hub. Alternatively, the LIAB can be connected directly to a host computer using a cross-link cable. Linux-device: `eth0`.
P403	4 pins USB B, female	USB Host: Universal Serial Bus (USB1.1) host. A high speed serial bus, which is standard on almost every modern computer. A variety of compatible devices exist, e.g. Web cameras, Wireless LAN adapters, and storage devices.
P402	4 pins USB A, female	USB Slave: Universal Serial Bus (USB1.1) slave. A high speed serial bus, which is standard on almost every modern computer. Enables the use of the LIABARM9200 as a USB slave device.
JP5	50 pins CF, male	CompactFlash® connector: CompactFlash® is a small removable mass storage device. They provide complete PCMCIA-ATA functionality.
JP6	50 pins Pin-head, male	Extension connector: The 50-pin connector on the LIABARM9200 standard baseboard connects directly to the JP1 connector on the LIABARM9200 microprocessor board. Please refer to section 2.2.2 for further information.
JP4	20 pins Pin-head, male	ARM Media connector: The 20-pin connector on the LIABARM9200 standard baseboard connects directly to the JP3 connector on the LIABARM9200 microprocessor board. Please refer to section 2.2.2 for further information.
JP401-402	4 pins Screw terms	Digital I/O connector: Four digital programmable input/output (PIO) pins. +3.3 V TTL compliant.

Table 2.2: *Connectors on the LIABARM9200 standard baseboard.*

3. Get your Board Up and Running

When delivered from LIAB ApS, your LIABARM9200 microprocessor board is preloaded with a boot loader and a Linux system. This system will boot when power is connected. In principle, you might apply power to the microprocessor board directly using the relevant pins on pin header JP2. However, in the following it is assumed that you have at hand both the LIABARM9200 microprocessor board and the standard baseboard.

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 LIABARM9200 from the protective, anti-static bags it was delivered in. In addition, the LIABARM9200 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 LIABARM9200 microprocessor board and one LIABARM-9200 standard baseboard.
- A serial cable. The end to be connected to the baseboard must be equipped with a female DB-9 connector. 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 7.5 to 24 volts, capable of delivering at least 4 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 LIABARM9200 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 LIABARM9200 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.

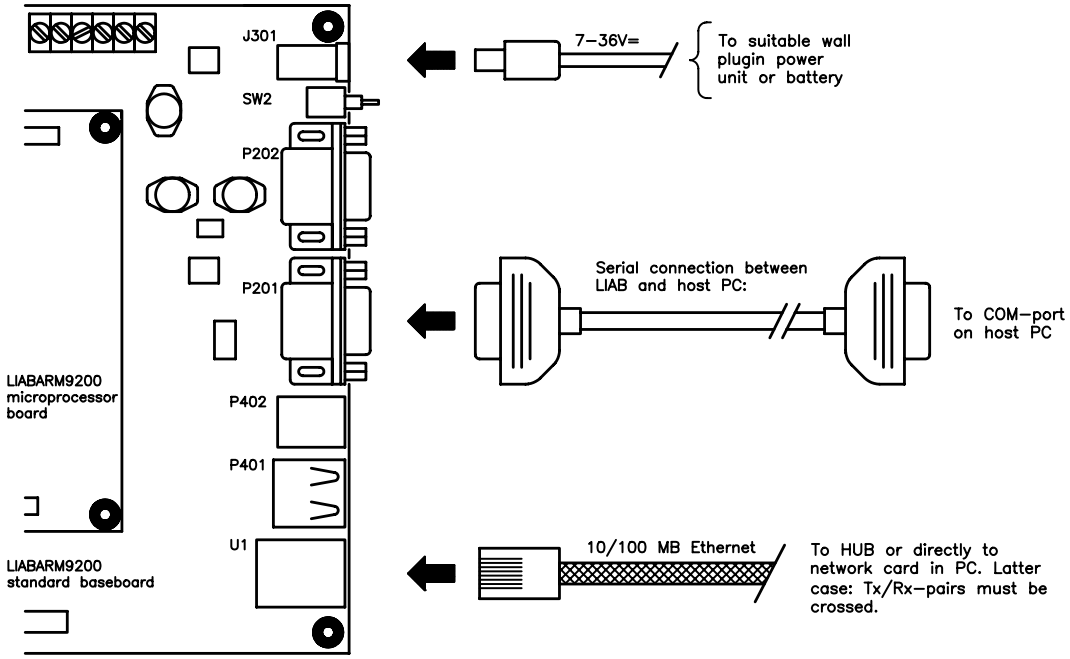


Figure 3.1: *Connections to the exterior world: power, serial port and network.*

3.2 Unpacking and Serial Connection

To operate the board for the first time, you need to mate the microprocessor board to the baseboard, aligning the three connectors as shown in Fig. 1.1 on page 9.

When assembled, connect the baseboard to a power source and to a host computer using the serial port connector `P201` (debug port) on the board. Later, when the network-related parameters have been configured using the serial port, you may connect the board to a 10/100 MBit Ethernet network using the RJ45 connector `U1`. The three mentioned connections to the exterior world are shown in Fig. 3.1.

A suitable serial cable can be purchased in a well-stocked computer shop. Alternatively, you can make one yourself if you have at hand two male DB-9 connectors and a length of cable with three wires. The wiring you need to make is shown in Fig. 3.2

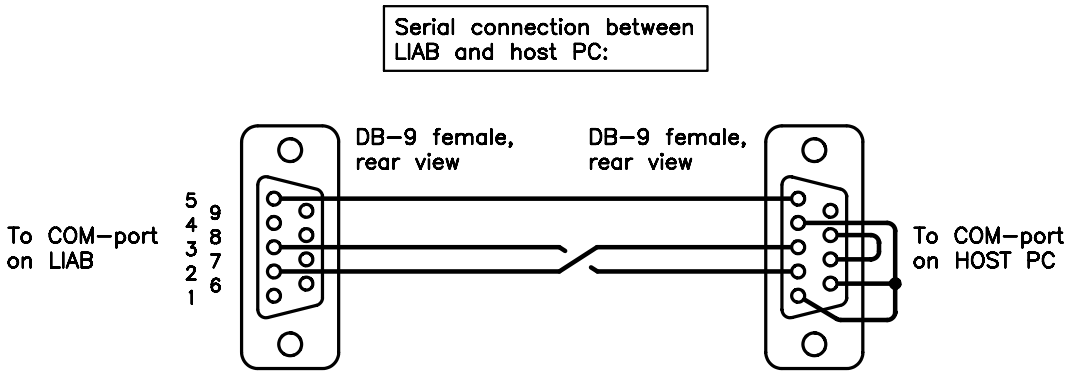


Figure 3.2: *Serial communication for the LIAB: connectors and wiring diagram.*

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 `J301`. A terminal emulator on the host computer must be started and configured for **115200 baud, 8 data-bits 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.
--------------o LIABARM (Linux In A Box) Bootloader o--------------

               -> Visit http://www.liab.dk <-

Release: 1.0, August 21, 2005 at 20:17 by midi
Copyright LIAB Electronics 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: **...
```

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 LIABARM (Linux In A Box) Bootloader o--------------

               -> Visit http://www.liab.dk <-

Release: 1.0, August 21, 2005 at 20:17 by midi
Copyright LIAB Electronics 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: **...
```

```
Resetting Ethernet PHY
MII PHYSID1,2 ....: 0x0181, 0xb881
PHY reset OK
-- Now Decompressing Image! ---- Now Decompresdecompressed image 15...
Putting tags at 20000100
Starting kernel ...
Linux version 2.4.25-vrs1 (root@msa) (gcc version 3.3.2) # Mon Sep ...
CPU: Arm920Tid(wb) revision 0
Machine: ATMEL AT91RM9200
On node 0 totalpages: 16384
zone(0): 16384 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: liabETH=00:90:82:FF:03:57 liabIP=192.168.1.180...
Calibrating delay loop... 88.47 BogoMIPS
Memory: 64MB = 64MB total
Memory: 58044KB available (1329K code, 245K data, 56K init)
Dentry cache hash table entries: 8192 (order: 4, 65536 bytes)
Inode cache hash table entries: 4096 (order: 3, 32768 bytes)
...
...
< many lines of Linux kernel initialization messages >
...
Starting cron...
Starting httpd...

LIAB distribution 4I
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 SW2 on the LIABARM-9200 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
 b <size> <initrd>: Setup for linux to boot, <size> is memory size ...
                    default 24 MB, <initrd> offset in bytes from .....
                    (for experts only, type "q <ret>" to boot ......
 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 0x100000
 l                : Load images using uuencoded data
*p                : Display and edit boot and network parameters
 q                : Quit monitor and continue boot procedure
 r <rate>         : Set baudrate (0:9600, 1:19200, 2:34800 3:57600
 s                : Scan memory for GZIP images
 x <addr>         : Load data to RAM, starting at address <addr>,
 z <src> <dst>    : Decompress GZIP image at <src> into <dst>
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 web-pages 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: liabETH=00:90:82:ff:03:5d
 2: liabIP=192.168.1.182,8,192.168.1.1
 3: liabHOST=liab2.liab.dk
Param>w
Do you want to write the parameters back to FPROM? [y/n]>y
Boot parameters start at: 0x007ff800
Param>
```

The parameter "liabETH=..." is the Ethernet MAC address. This should not be changed. The MAC address assigned to the LIAB can be found on the label on the bottom side of the board. 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 LIABARM9200 standard baseboard or you enter two successive "q" (quit) commands:

```
Param>q
Boot>q -- Now Decompressing Image! ---- Now Decompress
decompressed image 1498424 bytes
Putting tags at 20000100

Starting kernel ...
...
```

You will now observe a boot sequence similar to the one printed on page 20. 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 LIABARM9200 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 LIABARM9200 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 LIABARM9200 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 LIABARM9200 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 LIABARM9200 microprocessor board.

The sub-directory "`/hardware/baseboard`":
Schematics and component layout for the LIABARM9200 standard baseboard.

The directory "/software":
This directory contains software for the LIABARM9200 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 LIABARM9200 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 2.4.25 Linux kernel which is able to boot on the LIABARM9200 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 LIABARM9200 microprocessor board and LIABARM9200 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 LIABARM9200 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 LIABARM9200 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 LIAB-ARM9200 .

First, boot up your LIAB and use the program ftp to transfer the executable hello_arm to the LIABARM9200 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 LIABARM9200 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 LIABARM 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 LIABARM9200 microprocessor board includes a device driver for reading and controlling the state of the buttons and LED's located on the LIABARM9200 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 LIABARM9200 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 "`lamod`" is created.

```
root@liab# insmod LAmod
root@liab# mknod /dev/lamod 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 "LAmod" 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 "^V1" > /dev/lamod
root@liab# echo "^V2" > /dev/lamod
root@liab# echo "^V4" > /dev/lamod
root@liab# echo "^V8" > /dev/lamod
root@liab# echo "^V10" > /dev/lamod
...
root@liab# echo "^V40000" > /dev/lamod
```

You should now see that one segment or one LED light up, depending on the actual hex-value submitted to the driver. In order to display the number series "0...9" you might try:

```
root@liab# echo "^V3f" > /dev/lamod
< now try with ^V06, ^V5b, ^V4f, ^V66, ^V7d, ^V6d, ^V07, ^V7f, ^V6f >
```

It is also possible to read the state of both push buttons (SW1-2) and the HEX switch through the "LAmod" module. To read the state of the hex switch and the two push-buttons, we type:

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

The first two digits returned represents SW1 and SW2 (1 when button is depressed, 0 if not), and the latter represents the setting of the HEX switch.

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 Demo Program Using the LIABARM Module

Using the LIABARM module as described in the previous section is practical when testing and using a module from e.g. shell scripts. To use the module in a compiled C program, however, a series of POSIX standard function calls must be used. POSIX is an acronym for "Portable Operating System Interface", which is a set of open system standards based on UNIX. The relevant function calls in this demo program are "`open()`", "`close()`", "`read()`", and "`write()`".

The purpose of this demo program is to write a C program, capable of counting from 0-9 on the seven segment display located on the LIABARM9200 microprocessor board.

The first task in writing the program is to get a handle, a file descriptor, for the device-special-file we want to communicate with. A file descriptor can be created by 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 the LIABARM module is shown below.

```
int main()
{
    /* This is the file descriptor */
    int fd;
    /* Try to open the device in read/write mode */
    fd = open("/dev/lamod", O_RDWR);
    /* Check whether it opened correctly */
    if(fd < 0)
    {
        fprintf(stderr, "Error opening module\n");
        return -1;
    }

   .
   . Here all communication with the module is performed.
   .

   /* Close the device again*/
   close(fd);
}
```

The actual communication with the module is done using the file descriptor and the two function calls "`read()`" and "`write()`". The example below demonstrates how to count backwards or forwards on the seven segment display, depending on whether the HEX switch is on an equal or unequal number. The code snippet assumes that a valid file descriptor "`fd`" has been created, as described in the previous example.

```
int main()
{
    char str[40];
    int dir,i;
    int digits[10] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66,
                       0x6d, 0x7d, 0x07, 0x7f, 0x6f };

    < declaration of fd and opening of /dev/lamod as stated above >

    i = 0;
    dir = 1;

    while(1)
    {
        sprintf(str, "^V%02x", digits[i]);
        write(fd, str, 4);

        if(read(fd, str, 5) == 5)
            dir = strtol(&str[3], NULL, 16);

        if(dir%2)
          i--;
        else
          i++;

        if(i > 9)
          i = 0;
        if(i < 0)
          i = 9;
        sleep(1);
    }
}
```

For a complete code example, please consult appendix B and the "`demoprogram.c`" in the directory

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

on the accompanying CD-ROM.

In addition to the basic blocking read and write operations, described in this section, the module also supports non-blocking reads. An example of how to use this is provided in the example code "`lamoddemo.c`" located in the same directory.

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.

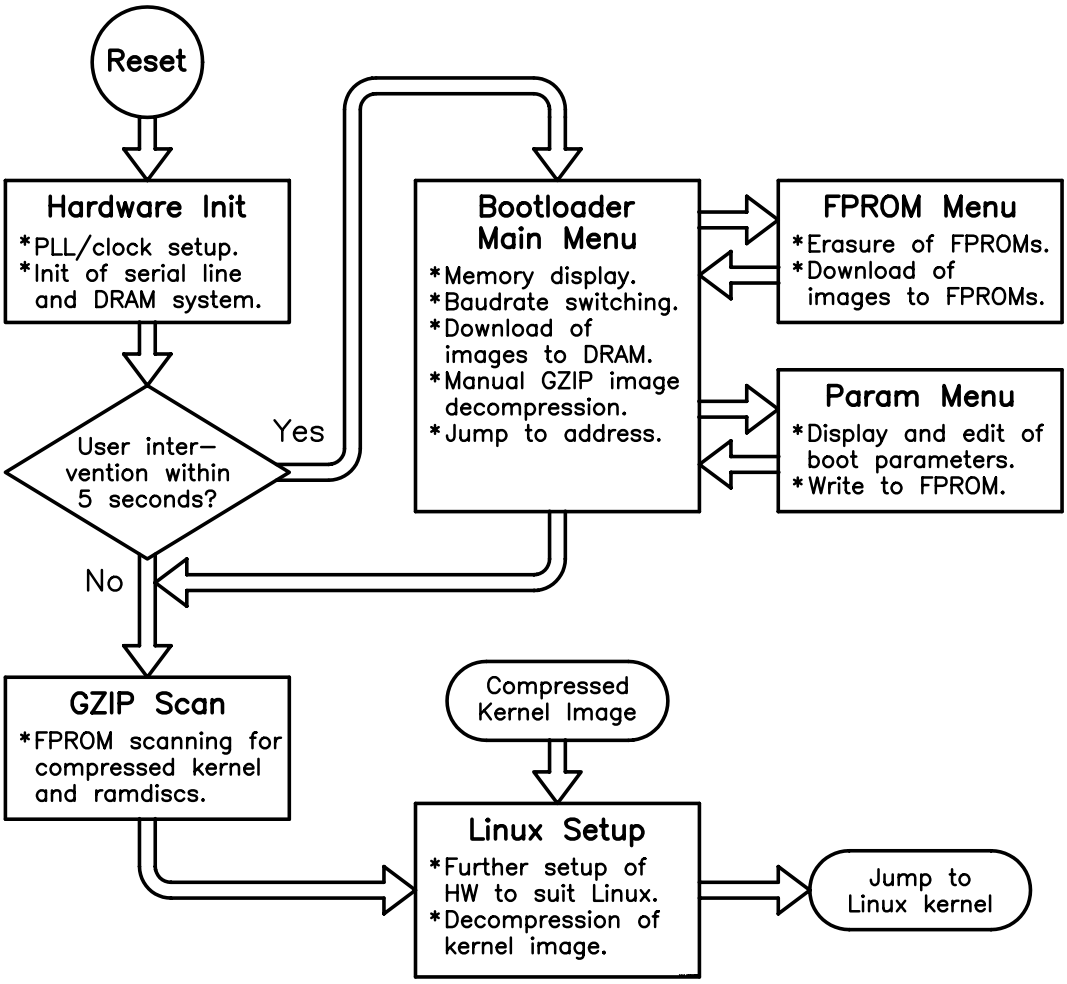


Figure 5.1: *The boot loader for the LIABARM9200 board.*

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. The JTAG interface

The JTAG (Joint Test Action Group) is an industry standard specification developed for PCB testing, which provides a means to test interconnects between chips on a board without using physical test probes. Chips with JTAG interface includes a so called boundary-scan cell for each pin on the device. This cell is able to capture a logic level from a pin or force a logic level onto that pin. Forced levels to a pin are serially shifted into the boundary-scan cell via the JTAG interface and captured levels are shifted out. The Atmel AT91RM9200 ARM processor includes boundary-scan cells on all digital pins. This gives the ability to emulate bus cycles via the JTAG interface, allowing access to peripheral devices such as the FLASH PROM.

The JTAG interface is located on the bottom side of the LIABARM9200 microprocessor board in the two columns of test point pads near JP1.

To identify the functions of the individual test points, please consult the schematics of the LIABARM9200 microprocessor board, located on the accompanying CD-ROM in the directory:

```
hardware/processorboard
```

If you accidentally erase the boot loader, you might need a JTAG programming device to restore it. In this case, please contact LIAB ApS.

7. 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.

Using the newer Linux-kernels (version 2.4.x or higher) this has become possible using the Linux kernel 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 LIABARM9200 , one get may two to three megabytes, depending of the hardware configuration of the LIABARM9200 microprocessor board, of FLASH disk for the storage of application-specific programs and data. The rest of this section deals with these systems on the LIABARM9200 board.

7.1 Memory Technology Devices, MTD

Note: before carrying out the procedures listed below, you should check if your LIABARM9200 already has support 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. After installation of the LIABARM9200 distribution on a PC with Linux one first copies a suitable kernel configuration file to the hidden file "`.config`". Next start "`make xconfig`"

```
user@host$ cd liab4I-ARM-sep-2005/software/liabkernel/linux
user@host$ cp liab_armAT91_net_mtd_jffs2.config .config
user@host$ make xconfig
```

In the main menu system, you may find: "`Memory Technology Devices, MTD`" where the configuration of the MTD system on a LIABARM9200 board can be investigated. LIAB has chosen to split the FPROM memory into five MTD partitions: "`Root`", "`JFFS2`", "`Boot`", "`Kernel`", and "`Param`". In the sub-item: "`Mapping drivers for chip access`" one can choose a suitable use of the five partitions, see Fig. 7.1:

1. The first megabyte contains the bootloader, and Linux kernel. These sections are placed on the MTD partitions "`Boot`" and "`Kernel`".

2. The partition "`Root`" is normally used for the compressed initial ramdisk-image, which is decompressed and transferred to the first ramdisk during boot. This disk constitutes the root disk for the Linux system. Thus, one cannot use this partition for a JFFS2 file system. The size of this partition must be at least 3456 kilobytes.

3. The partition "`JFFS2`" is used for the JFFS2 file system. The upper limit of the size is determined by the amount of FPROM.

4. At the top of the FPROM, 64 kilobytes, are reserved for bootstrap and boot parameters. The corresponding MTD-partition is "`Param`".

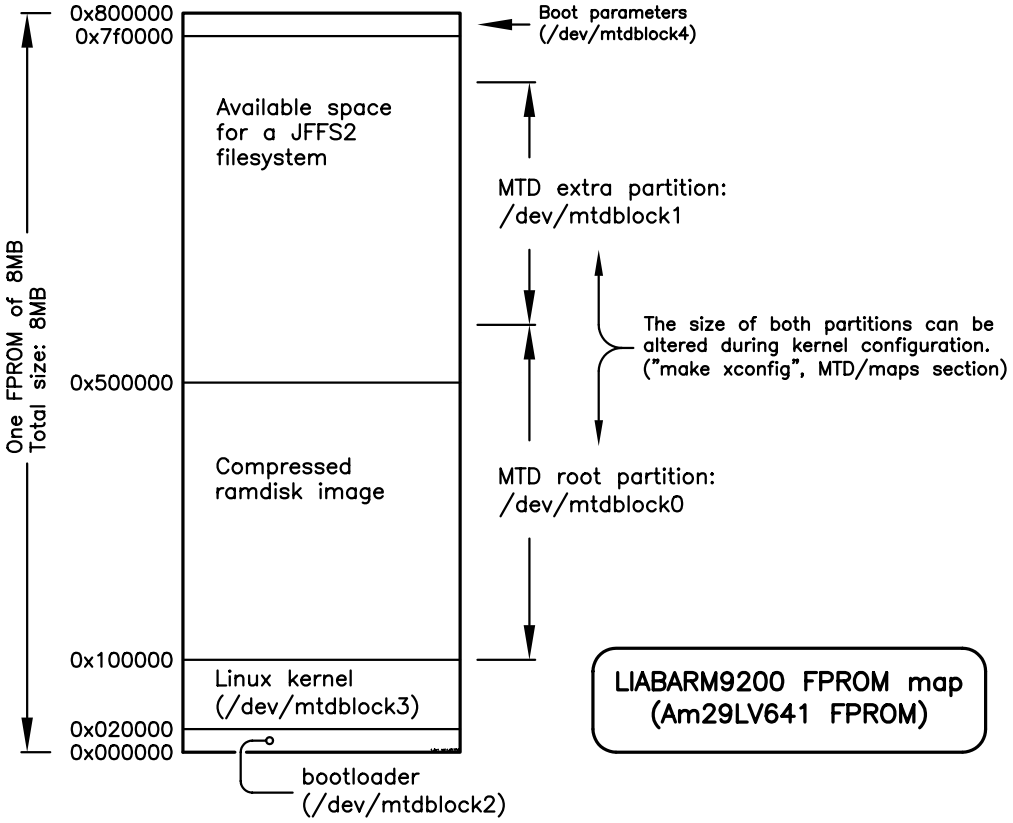


Figure 7.1: *Layout of FPROM memory on a standard LIABARM9200 .*

The sizes of the partitions must be a multiple of 64 kilobytes and the sum of them must be less than or equal to $8192 - 1024 - 128 - 64 = 6976$ kilobytes. 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)
Jffs2        /dev/mtd1 (90/1)   /dev/mtdblock1 (31/1)
Boot         /dev/mtd2 (90/2)   /dev/mtdblock2 (31/2)
Kernel       /dev/mtd3 (90/3)   /dev/mtdblock3 (31/3)
Param        /dev/mtd4 (90/4)   /dev/mtdblock4 (31/4)
```

When a suitable partitioning has been chosen, click "`Save and Exit`" in the main menu and build a new kernel by issuing:

```
user@host$ make clean dep vmlinux
user@host$ cd ../../liabboot
user@host$ mkkernel
```

Now a stripped and compressed Linux kernel suitable for the LIABARM9200 has been created under the name: "`vmlinux.bin.gz`". This file can be uploaded

using `ftp` or `rcp` to a running LIABARM9200 . Finally, the kernel is written to FPROM using the MTD partitions (it is assumed that the your LIABARM9200 has the hostname "`liab`"):

```
user@host$ ftp liab
< Log on to the LIABARM as root >
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put vmlinux.bin.gz
...
ftp> quit
user@host$ telnet liab
Trying 192.168.1.180...
Connected to liab
< log on again as root >
root@liab# ls
vmlinux.bin.gz
root@liab# dd conv=sync if=vmlinux.bin.gz of=/dev/mtdblock0 bs=1M count=4
< wait until the #-prompt returns >
root@liab#
```

As an alternative, one may upload a uuencoded version of the compressed Linux kernel to the FPROM using the boot loader. The uuencoded file is named "`v`".

I the directory "`software/liabkernel/kernelimages`" precompiled kernels with MTD- and JFFS2-support may be found. These images can be uploaded and written to the FPROM using the procedure described above.

7.2 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`".

7.3 A Ramdisk having support for MTD and JFFS2

The initial ramdisk in the `liab4I-ARM-sep-2005` distribution for the LIABARM9200 microprocessor board includes support for the mounting of the "`JFFS2`"-partition in the FPROMs as an `JFFS2` file system over a specified mount point. To initialize a `JFFS2` file system you simply erase the area of the FPROMs that corresponds to the "`JFFS2`"-partition!

Press the reset-button and let the LIABARM9200 board make a full Linux boot. Browse back in the console output and find the information about the partitioning of MTD system:

```
...
liab flash device: 1000000 at 10000000
 Amd/Fujitsu Extended Query Table v1.1 at 0x0040
number of CFI chips: 1
Creating 5 MTD partitions on "Physically mapped flash":
0x00100000-0x00500000 : "LIABARMAT91 FLASH Root partition"
0x00500000-0x007f0000 : "LIABARMAT91 FLASH JFFS2 partition"
0x00000000-0x00020000 : "LIABARMAT91 FLASH Boot partition"
0x00020000-0x00100000 : "LIABARMAT91 FLASH Kernel partition"
0x007f0000-0x00800000 : "LIABARMAT91 FLASH Param partition"
...
```

First, press the reset-button again and gain access to the boot loader by sending three dots to the console on the COM1-port. Second, erase the "`JFFS2`"-partition using the "`f`" (Flash) sub-menu in the boot loader. Third, under the "`p`" (Parameter) sub-menu you may insert a parameter: "`liabJFFS2=<dir>`", indicating to the Linux system which directory to be used as mount point for the JFFS2 filesystem. Last, you may add a parameter: "`liabRUN=<program or script>`" which indicates a startup program or script to be executed at boot time:

```
-------------o Linux In A Box (LIAB) Bootloader o-------------
               -> Visit http://www.liab.dk <-
...
```

```
THE BOOT LOADER, YOU MUST SEND 3 DOTS WITHIN THE NEXT 5 SECONDS: *
LIAB bootloader, 'h' for help
Boot>f
Entering FPROM utility submenu, 'h' for help
Memory configuration of LIAB
AMD flash at addr 0x00000000 to 0x007fffff
FPROM>e 500000 7effff  << NB:  7effff = 7f0000 - 1 !>>
...
Do you want to erase? [y/n]>y
FPROM>q
Boot>p
Entering the boot/network parameter editor, 'h' for help,
use options 'n' and 'a' for editing of network stuff.
Param>p
 1: liabIP=192.168.1.180,8,192.168.1.1
 2: liabHOST=liab.liab.dk
Param>i 3 liabJFFS2=/jffs2
Param>i 4 liabRUN=/jffs2/StartApplication
Param>w
Do you want to write the parameters back to FPROM? [y/n]>y
Param>
< press reset and let the LIABARM boot to a login-prompt >
```

The first time the LIABARM9200 boots and the "JFFS2"-partition is mounted, no file system on partition exists. However, the proper initialization starts when you try to create a file:

```
root@liab# echo "LIABARM" > tmpfile
root@liab# rm tmpfile
root@liab# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/ram0              15M  7.5M  7.1M  52% /
/dev/mtdblock1        3.0M  324K  2.7M  11% /jffs2
```

Please note that even if the JFFS2 file system is known to be empty, the output from the "df" command indicates that 320 kilobytes are used! This is because the file system reserves a number of FPROM sectors as work space.

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. Welsh and L. Kaufman, *"Running Linux"*, O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 1996, ISBN 1-56592-151-8. 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] M. J. Bach, *"The Design of the UNIX Operating System"*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986, ISBN 0-13-201757-1. 8

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

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

[8] 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 LIABARM9200 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 LIABARM9200 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 LIABARM9200 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. Demo program

This demo program demonstrates the basic features of the LIABARM module, described in section 4.6 on page 31.

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

int main()
{
  /* This is the file descriptor */
  int fd;
  /* Number of bytes read */
  int nobytes;
  /* Buffer to hold characters to write and those read */
  char str[40];
  int dir,i, digits[10] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66,
                            0x6d, 0x7d, 0x07, 0x7f, 0x6f };

  /* Try to open the device in read/write mode */
  fd = open("/dev/lamod", O_RDWR);

  /* Check whether it opened correctly */
  if(fd < 0)
    {
      printf("Error opening module\n");
      exit(0);
    }
```

```
    i = 0;
    dir = 1;
    while(1)
    {
        sprintf(str, "^V%02x", digits[i]);
        write(fd, str, 4);

        if(read(fd, str, 5) == 5)
          dir = strtol(&str[3], NULL, 16);

        if(dir%2)
          i--;
        else
          i++;

        if (i > 9)
          i = 0;
        if (i < 0)
          i = 9;
        sleep(1);
    }


  /* Close the device again*/
  close(fd);

  return 0;
}
```

C. Schematics and Component Layout

On the following pages the complete schematics and component placements for the LIABARM9200 microprocessor board and the standard baseboard are presented.

Figure no.	Description:
C.1	LIABARM9200 : Block diagram.
C.2	LIABARM9200 : AT91RM9200 ARM microprocessor, DEBUG/JTAG test points , JP1-3 connectors, bus transceivers.
C.3	LIABARM9200 : EEPROM, DRAM, and FLASH subsystems.
C.4	LIABARM9200 : Ethernet PHY subsystem.
C.5	LIABARM9200 : Power supply and decoupling.
C.6	LIABARM9200 : Human IO, 7-segment, LED, switches.
C.7	LIABARM9200 : Component placement, top.
C.8	LIABARM9200 : Component placement, bottom.
C.9	Standard Baseboard: Block diagram.
C.10	Standard Baseboard: Connectors, JP1, JP5, JP6. RS232 debug console driver.
C.11	Standard Baseboard: +5.0 V power supply.
C.12	Standard Baseboard: I/O ports, USB Host power controller.
C.13	Standard Baseboard: Component placement, top.

Table C.1: *Overview of schematics and component placements on the following pages.*

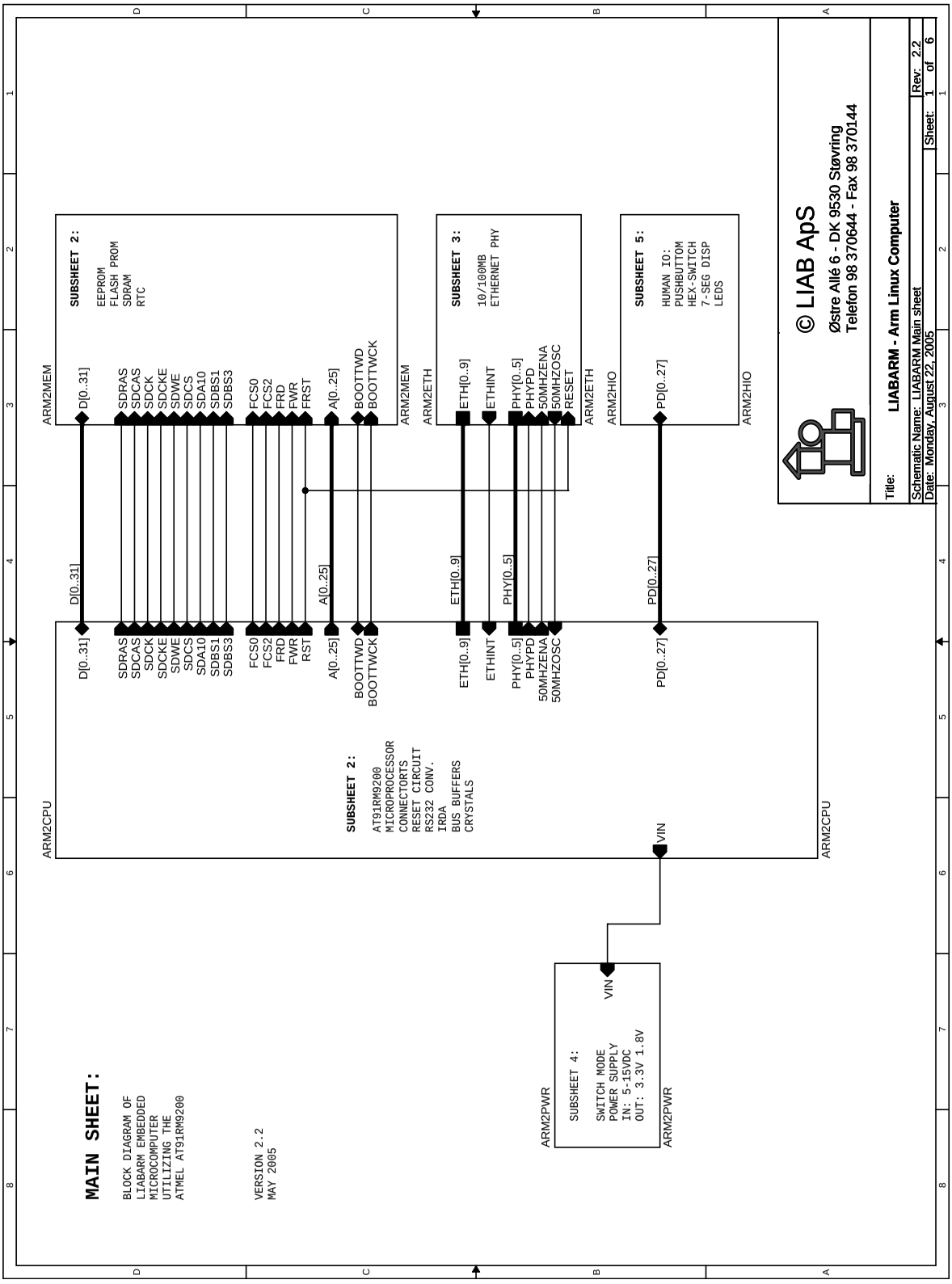


Figure C.1: *LIABARM9200 : Schematics page 1: Block diagram.*

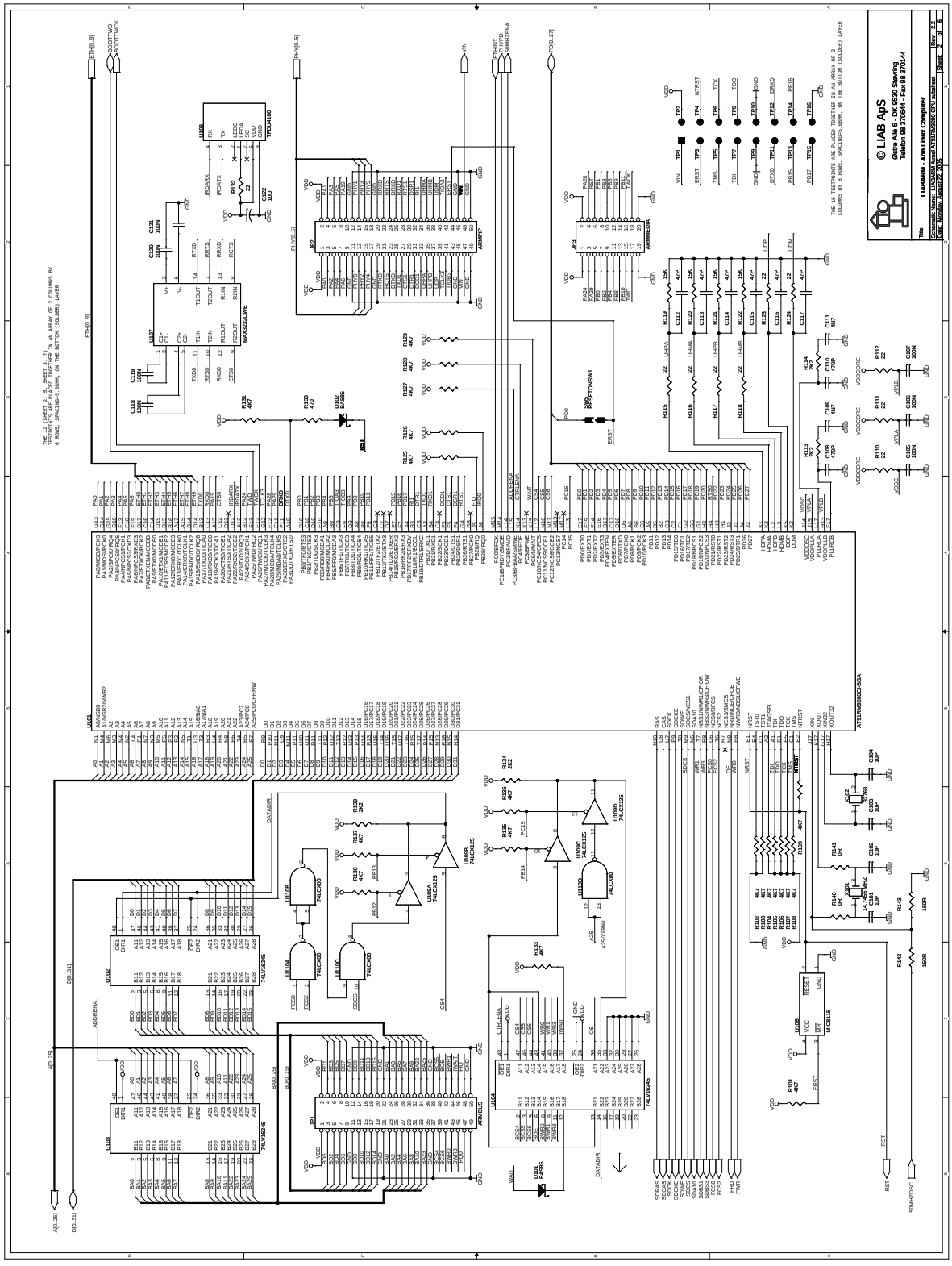


Figure C.2: *LIABARM9200 : Schematics page 2: AT91RM9200 ARM microprocessor, DEBUG/JTAG test points , JP1-3 connectors, bus transceivers. Overview of schematics and component placements on the following pages.*

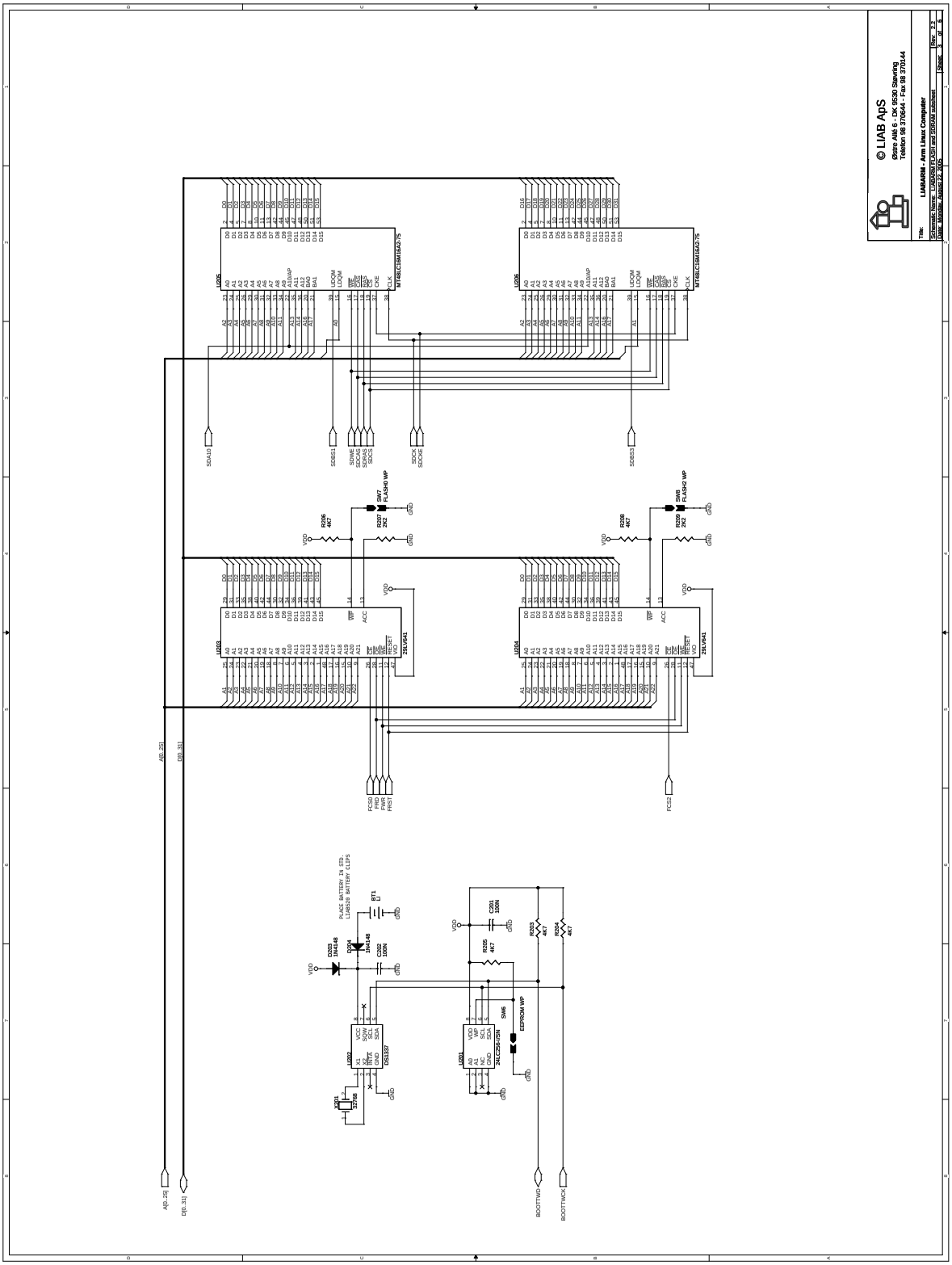


Figure C.3: *LIABARM9200 : Schematics page 3: EEPROM, DRAM, and FLASH subsystems.*

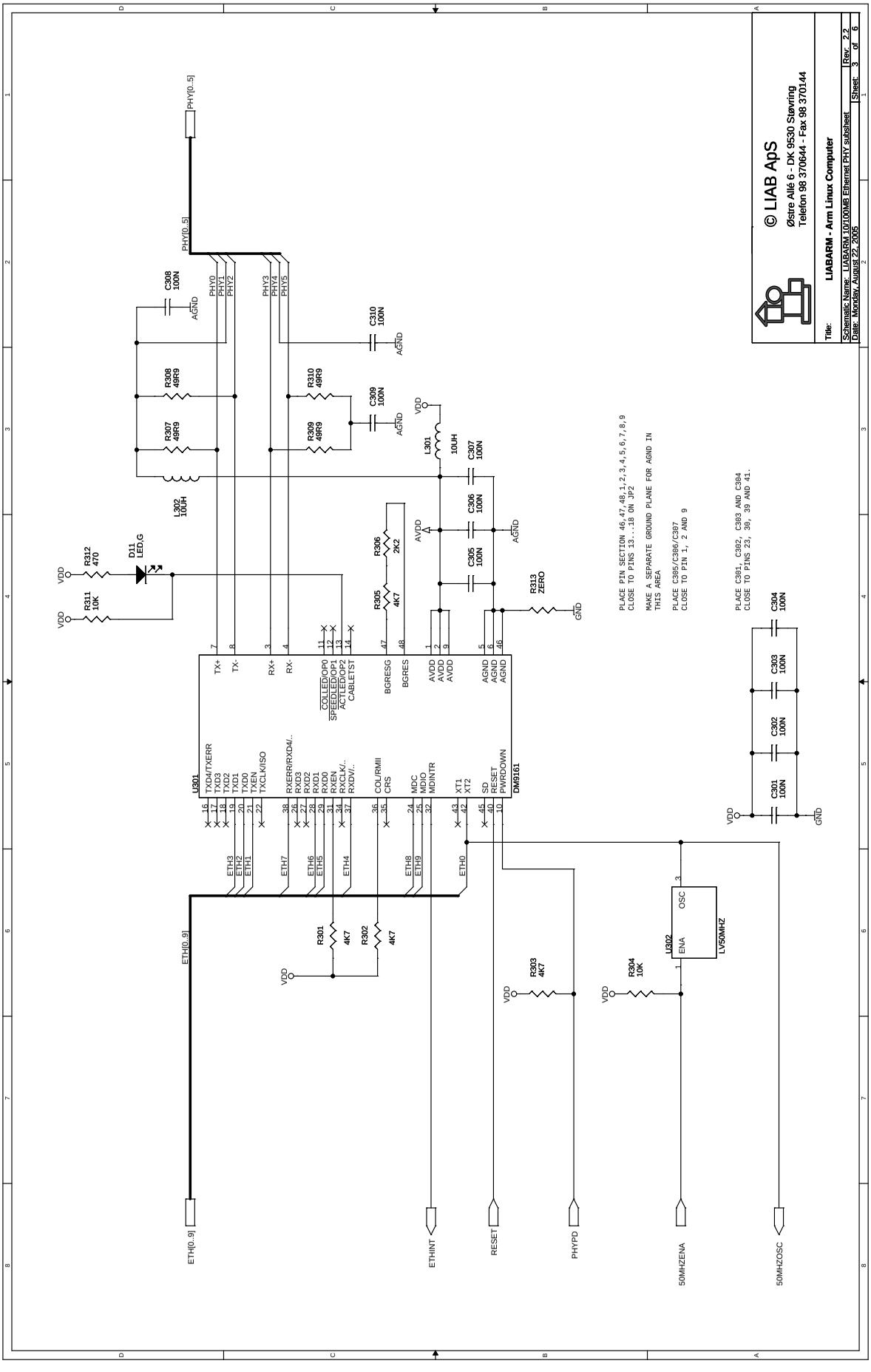


Figure C.4: *LIABARM9200 : Schematics page 4: Ethernet PHY subsystem.*

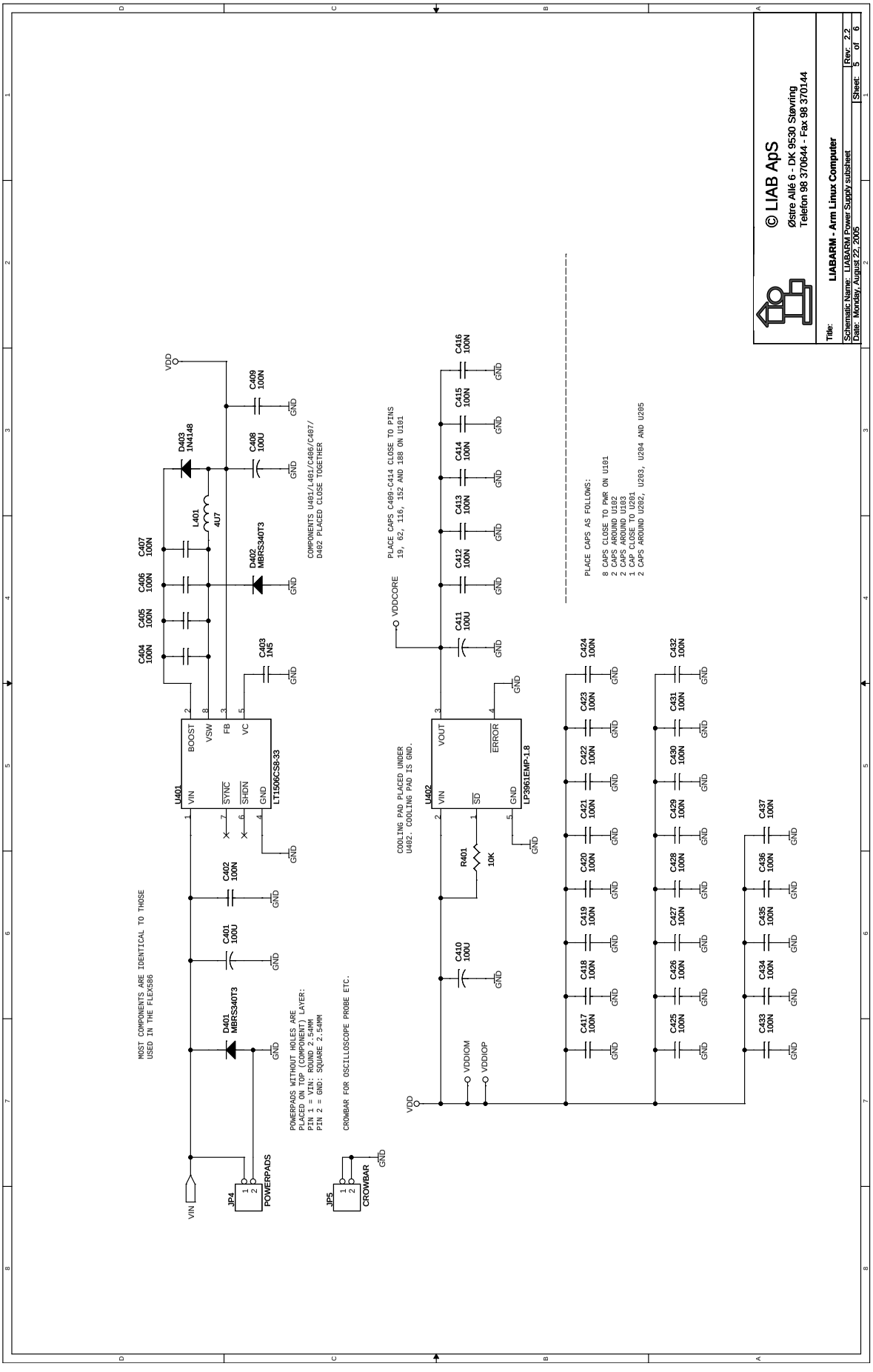


Figure C.5: *LIABARM9200 : Schematics page 5: Power supply and decoupling.*

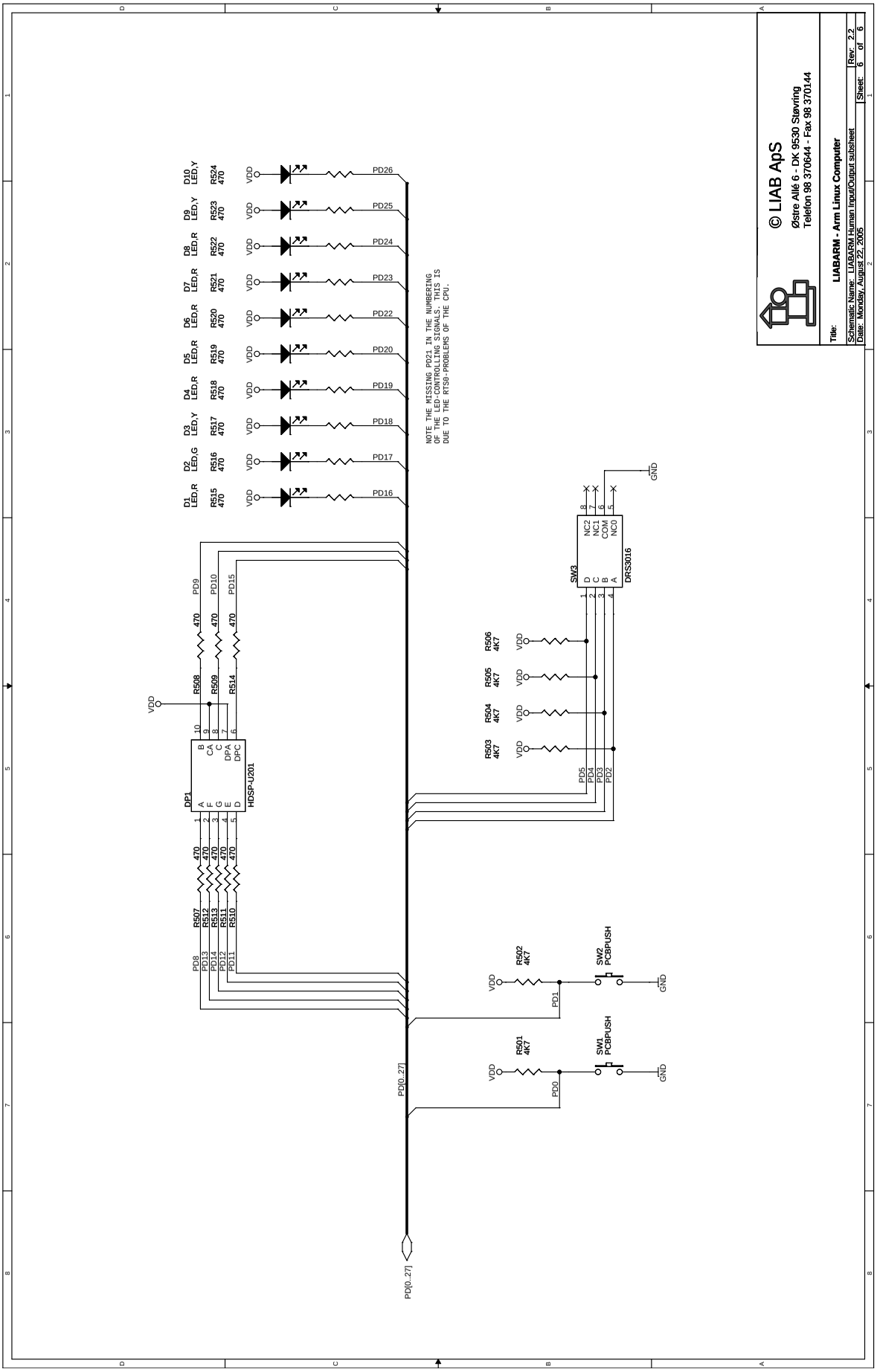


Figure C.6: *LIABARM9200 : Schematics page 6: Human IO, 7-segment, LED, switches.*

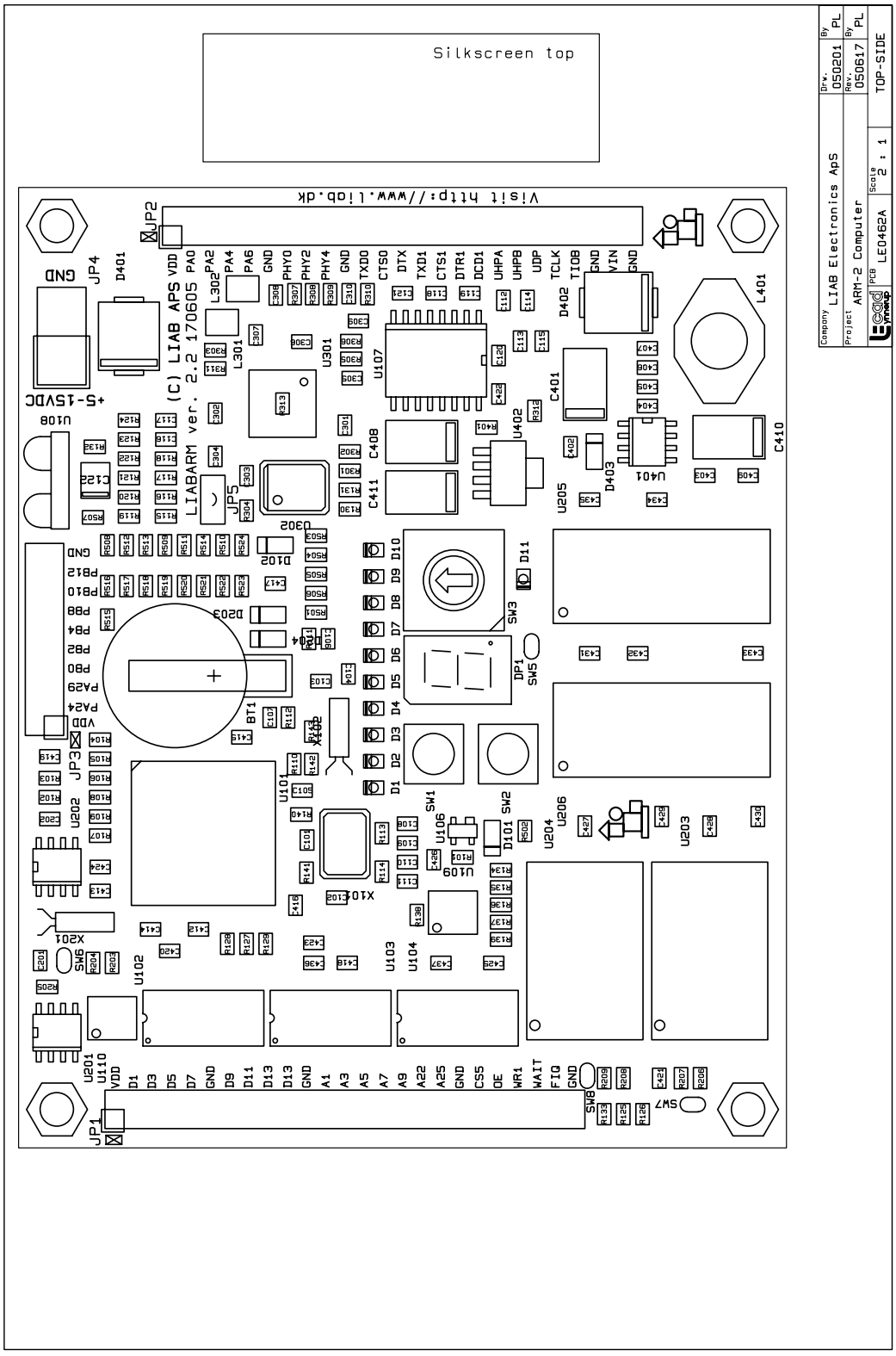


Figure C.7: *LIABARM9200 : Component placement page 1: Top.*

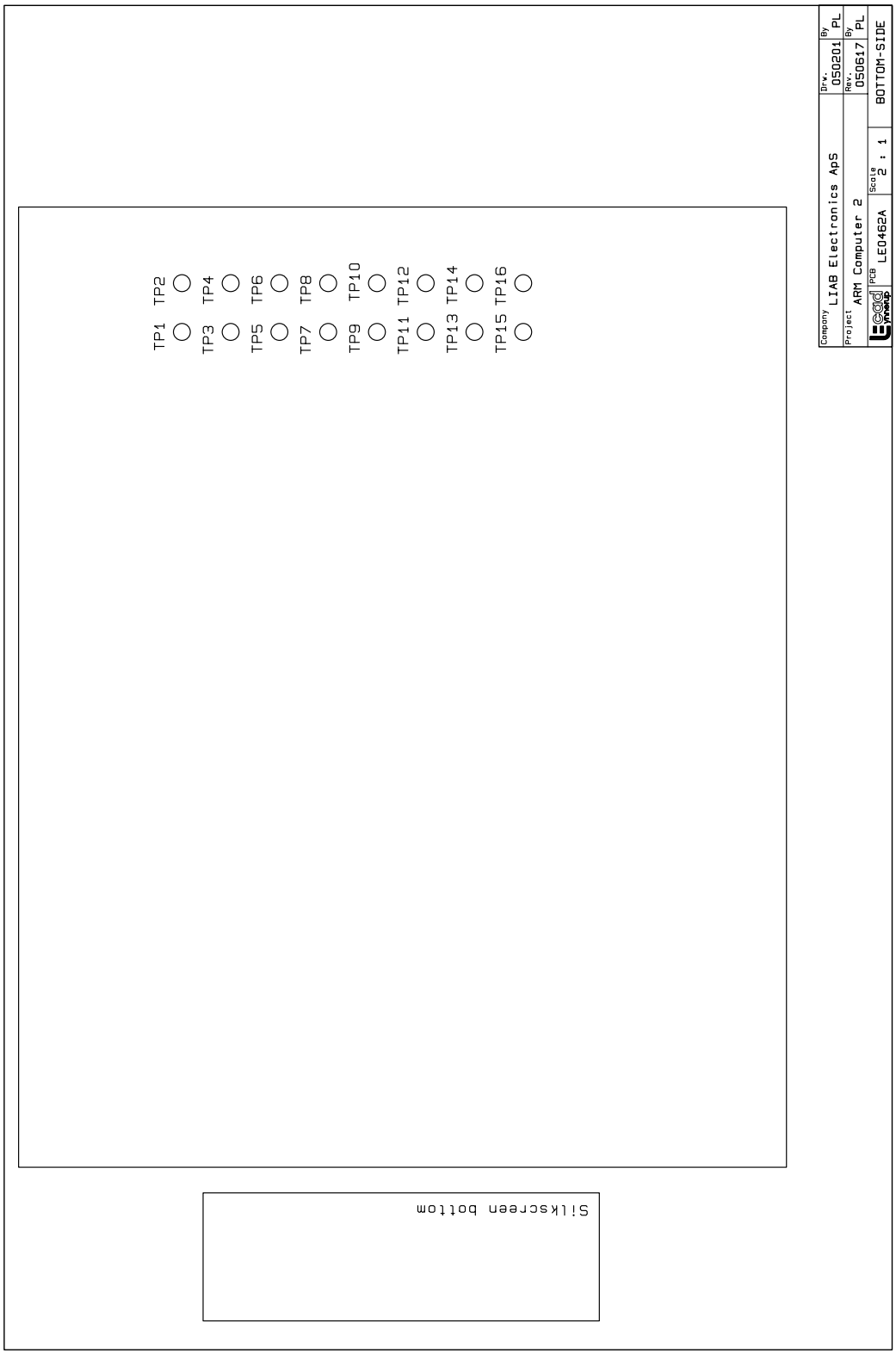


Figure C.8: *LIABARM9200 : Component placement page 2: Bottom.*

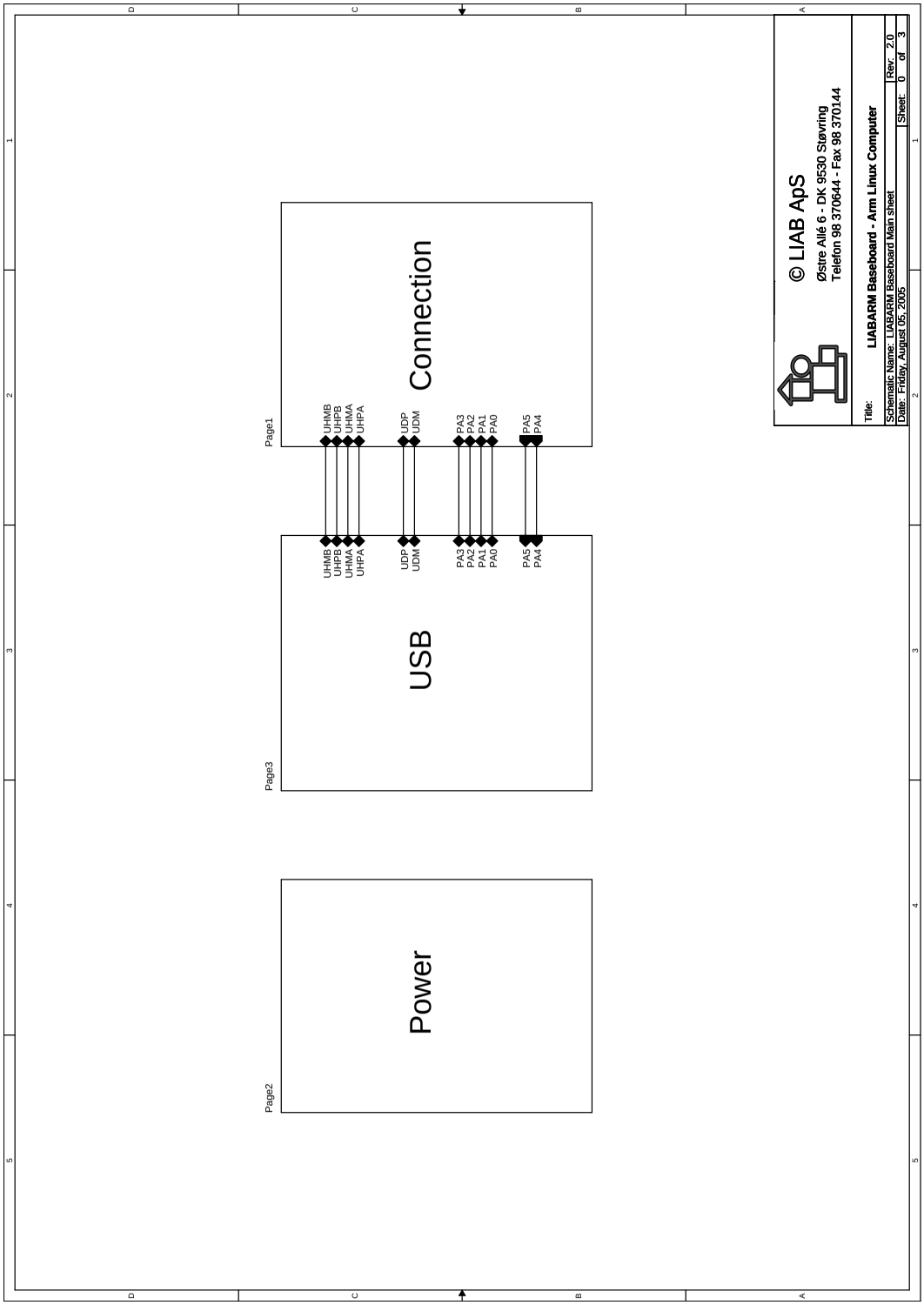


Figure C.9: *Standard Baseboard: Schematics page 1: Block diagram.*

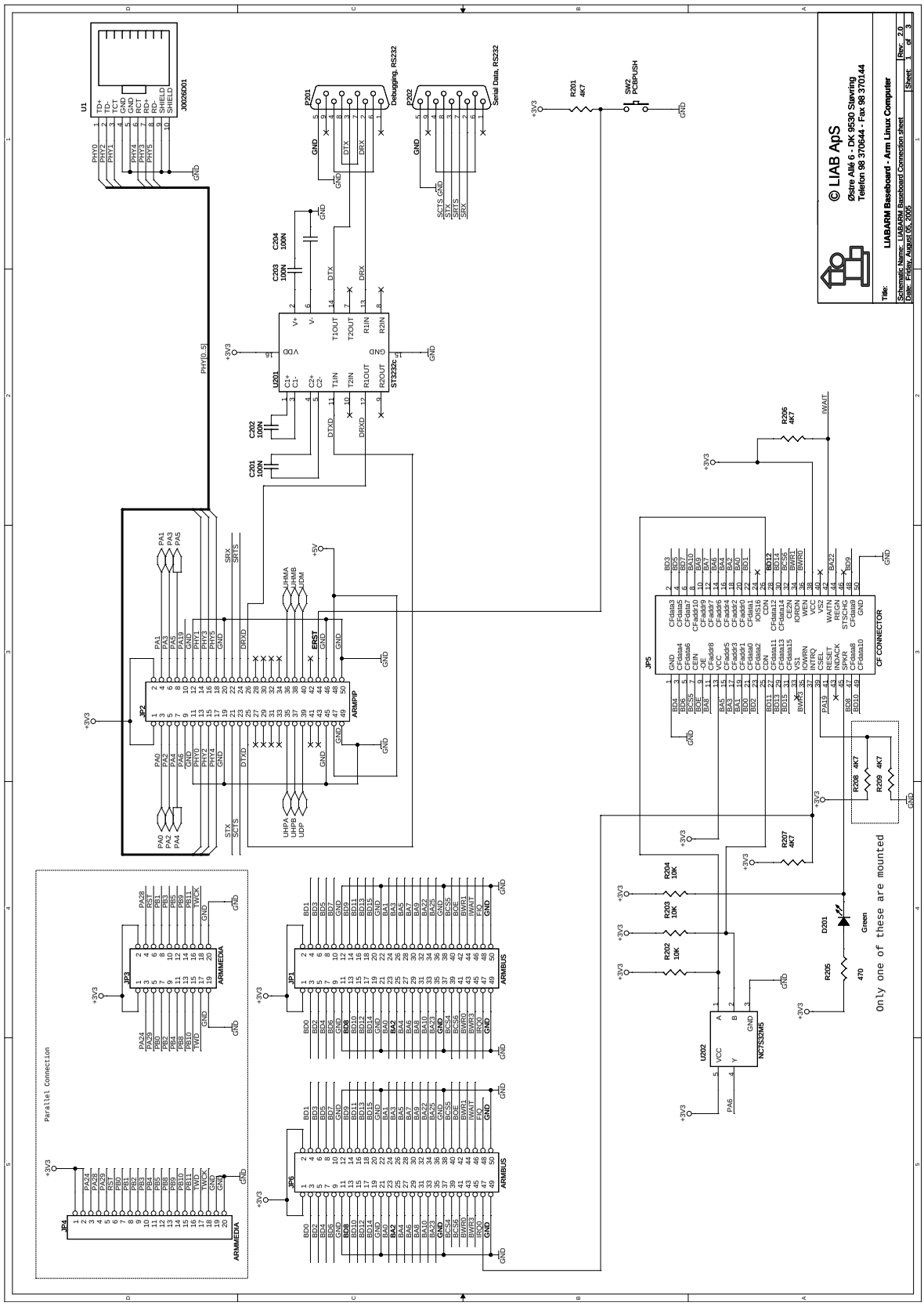


Figure C.10: *Standard Baseboard: Schematics page 2: Connectors, JP1, JP5, JP6. RS232 debug console driver.*

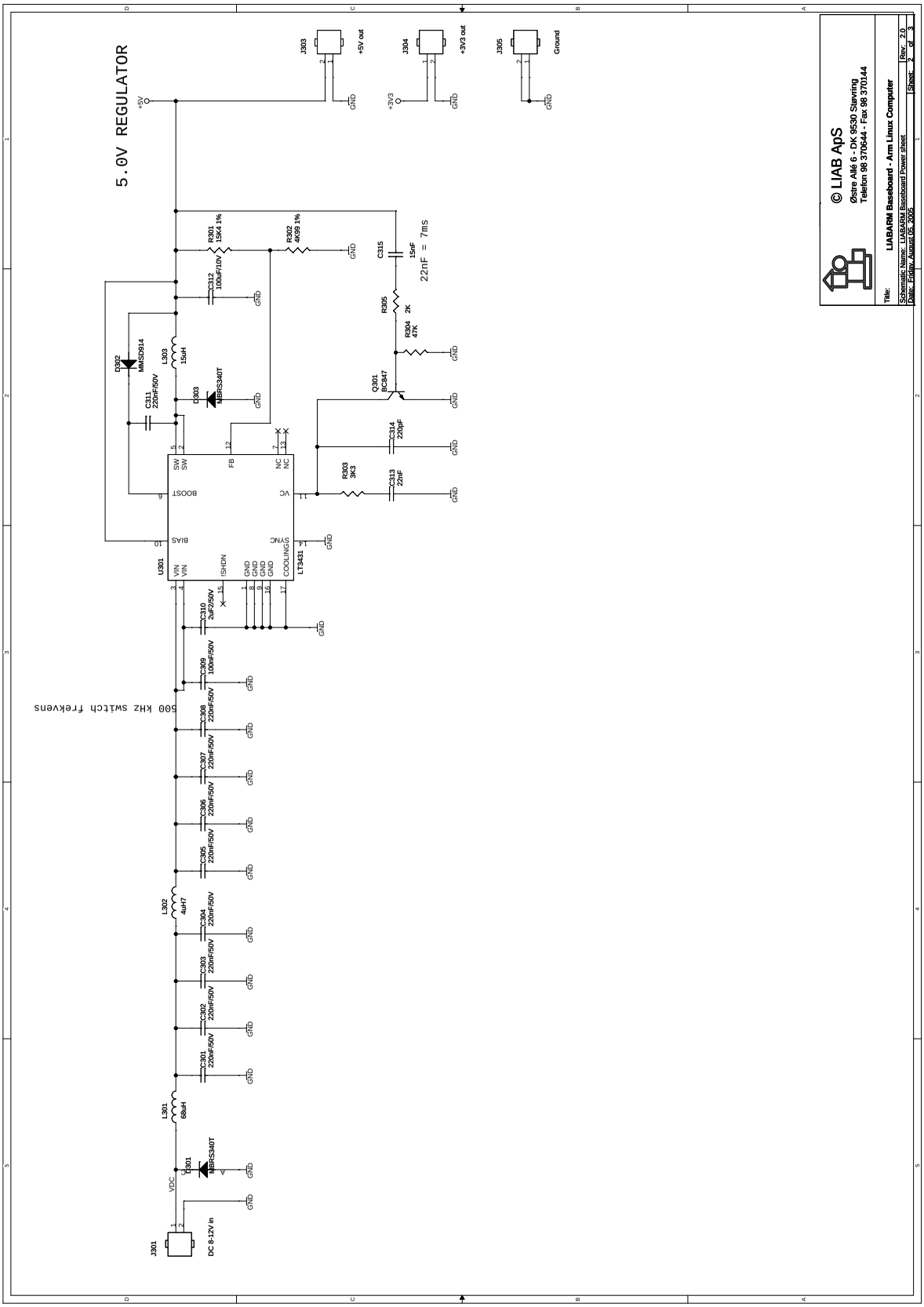


Figure C.11: *Standard Baseboard: Schematics page 3: +5.0 V power supply.*

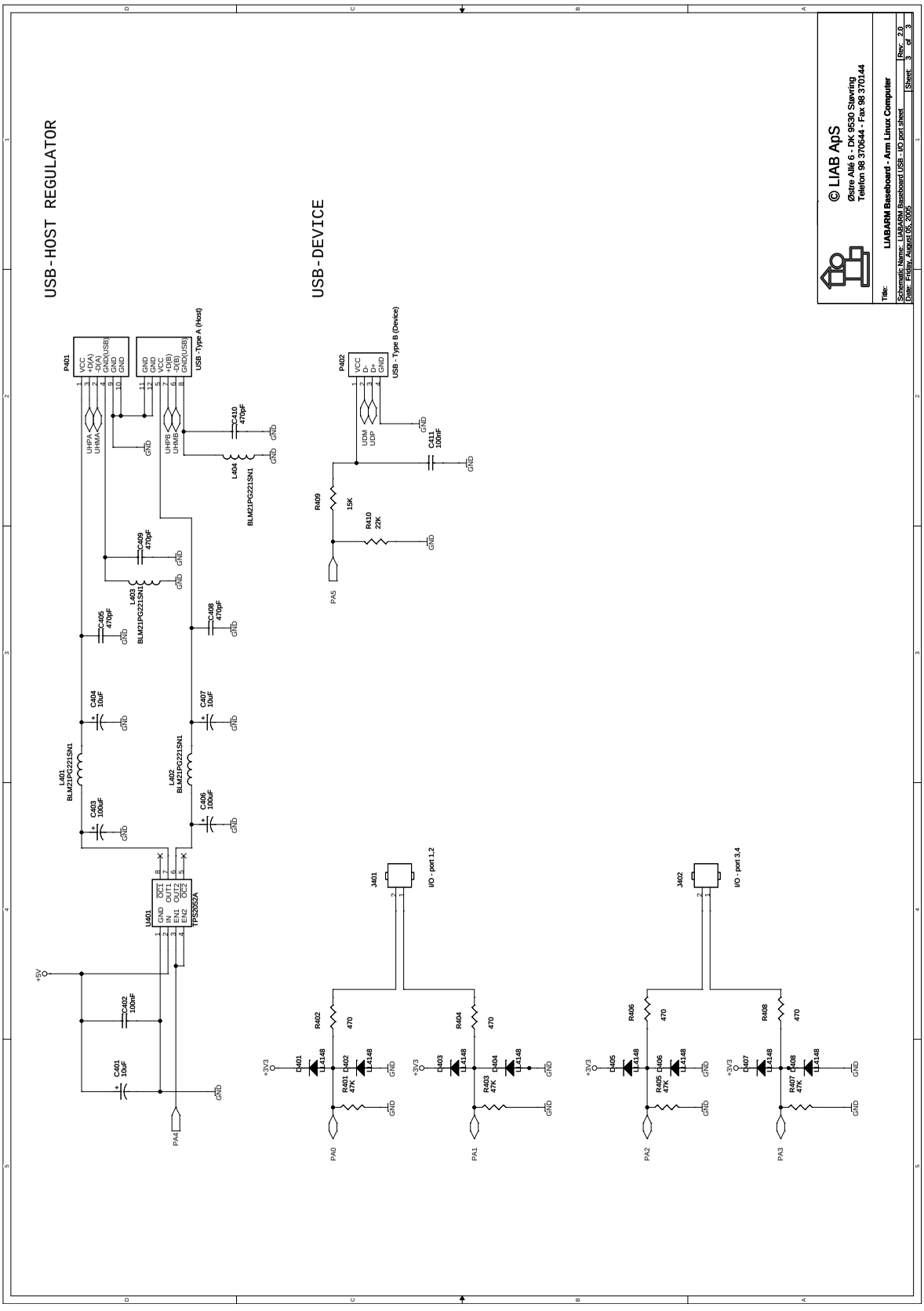


Figure C.12: *Standard Baseboard: Schematics page 4: I/O ports, USB Host power controller.*

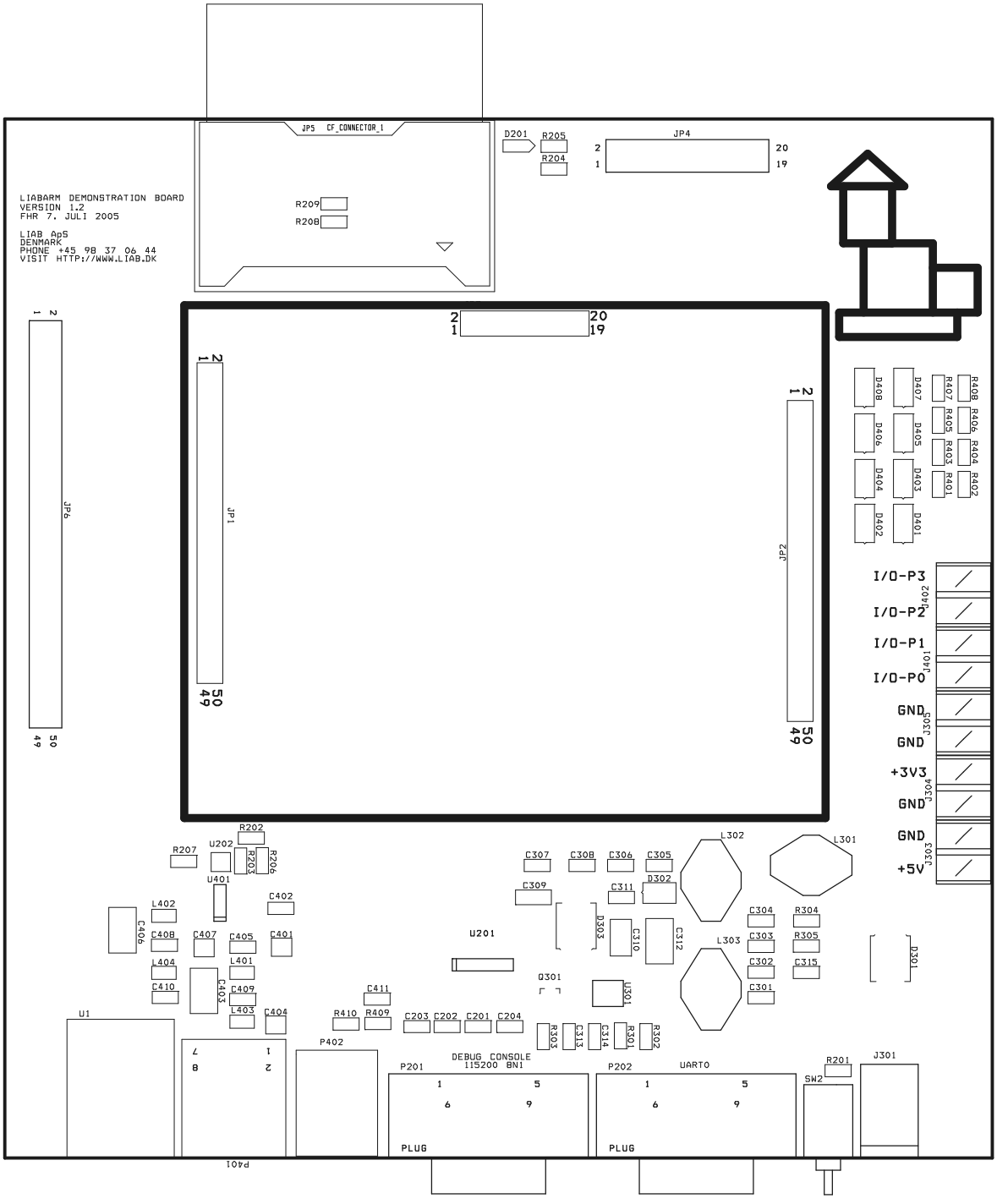


Figure C.13: *Standard Baseboard: Component placement page 1: Top.*