

MMnet1000 Linux System

User's Guide

Version 20090601

PROPOX®
Many ideas one solution

Contents

1. Introduction.....	3
2. Quick start guide	4
3. Boot sequence and system components	7
4. Arrangement of system components in NAND Flash.....	12
5. Software upgrade	13
6. Peripherals handling.....	15
7. Software Development Kit and example applications	23
8. System compilation	24
9. FAQ.....	26
10. Useful links	27
11. License, warranty and technical support	27
12. Attachments	28

1. Introduction

This document describe Linux system and software delivered with MMne1001 and MMnet1002 modules. It is applicable to 20090530 version of software, if you have older version, please upgrade. Newest version can be downloaded from:

<http://www.propox.com/download/software/MMnet1000-CD-20090601.zip>

Software version 20090601 is based on:

- U-Boot-2009.01
- linux-2.6.29.3
- OpenWrt KAMIKAZE r13340

Before start working, you should also read hardware related documents:

- MMnet1001: http://www.propox.com/download/docs/MMnet1001_en.pdf
- MMnet1002: http://www.propox.com/download/docs/MMnet1002_en.pdf

2. Quick start guide

2.1. Connections

A. MMnet1001 connections

1. Connect RS232 DBGU port of module to PC with use of level converter (you can use RS232 port on EVBmmTm evaluation board).
2. Connect to module regulated 3.3V DC Power supply with 500mA capability. Pay particular attention to this connection because module is not protected from inverse polarity nor overvoltage.
3. Connect module to local Ethernet Network (this is optional).

Example of connections on EVBmmTm evaluation board:

Interface	EVBmmTm peripheral	Module socket	Connection
Power	+3.3V	C1	Necessary
	GND	D1	
RS232 – DBGU	RS232_1 RxD	A7	
	RS232_1 TxD	B7	
Reset	RESET	A16	Optional
USB Device	USB DP	B12	
	USB DM	A12	
USB Host Top	HOST_USB T_D+	B10	
	HOST_USB T_D-	A10	
USB Host Bottom	HOST_USB D_D+	B11	
	HOST_USB D_D-	A11	
RS232 – UART0	RS232_2 RxD	B5	
	RS232_2 TxD	A5	
SD/MMC	CARD CS	TBD.	
	CARD MOSI	TBD.	
	CARD CLK	TBD.	
	CARD MISO	TBD.	
	CARD INS	TBD.	
	CARD UNL	TBD.	

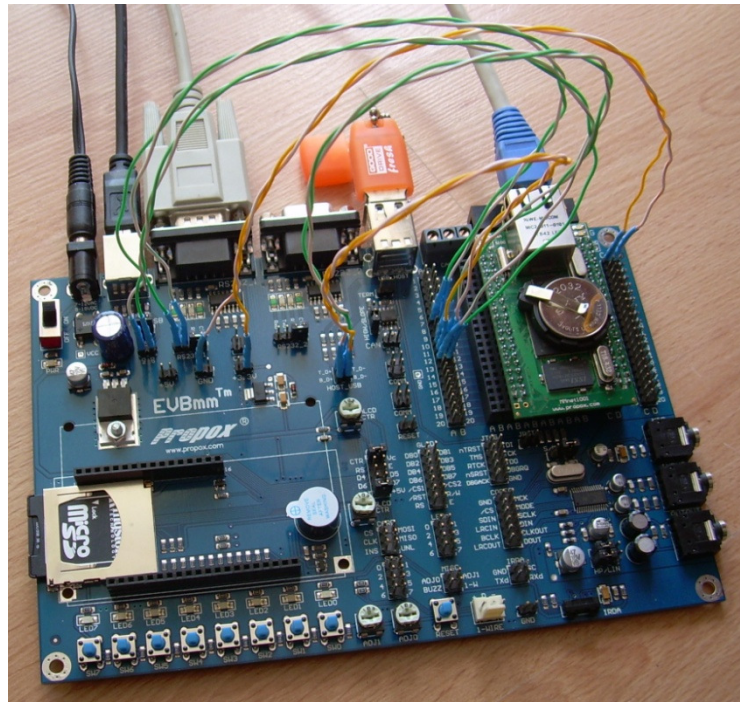


Figure 1 MMnet1001 module on EVBmmTm evaluation board.

B. MMnet1002 connections

1. Connect null modem serial cable between module serial port P0 and PC. Jumpers on module should be in „DBGU” position.
2. Connect 8 – 24V DC (ground on connector’s shield) power supply to module. Module is protected from inverse polarity.
3. Connect module to local Ethernet Network (this is optional).

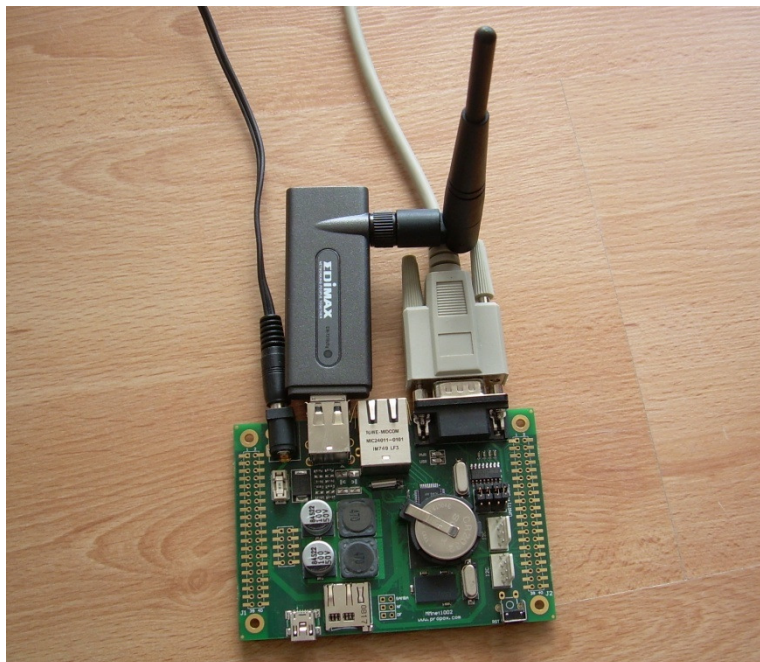
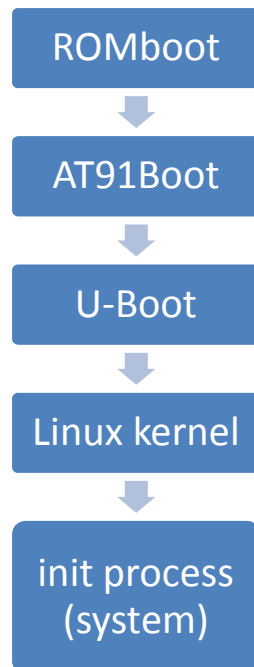


Figure 2 MMnet1002 module with USB Wi-Fi card.

3. Boot sequence and system components

3.1. System start-up

Several pieces of software are involved in booting Linux system on module. This process is described below:



3.2.ROMboot

This is a program in microprocessor's ROM memory. Detailed description of it can be found in MCU documentation in chapter 13 „AT91SAM9260 Boot Program”.

On MMnet1001/1002 modules ROMboot load AT91Boot program from first 4096 bytes of NAND Flash memory to MCU internal SRAM memory and launch it.

3.3. AT91Boot

AT91Boot is a first stage bootloader. Its purpose is to initialize system and to load second stage bootloader. Source code of AT91Boot adapted to MMnet1001/1002 modules can be found on CD in file /src/Bootstrap-v1.10-MMnet1000.tar. Documentation is contained in this archive in doc directory.

AT91Boot configured for MMnet1001/1002 modules load program from NAND Flash memory address 0x00040000 to SDRAM memory address 0x23F00000 and launch it.

3.4. U-Boot

U-Boot is a second stage bootloader. Its purpose is to load and launch operating system or other software that is intended to run on module. It has a lot of options, allow to load application from NANDFlash/DataFlash memory, from TFTP server through Ethernet, from external USB storage, can program Flash memory and more. Detailed documentation can be found at address: <http://www.denx.de/wiki/U-Boot/Documentation>. Below is a list of supported commands:

```
U-Boot> help
?      - alias for 'help'
base   - print or set address offset
boot   - boot default, i.e., run 'bootcmd'
bootd  - boot default, i.e., run 'bootcmd'
bootm  - boot application image from memory
bootp  - boot image via network using BOOTP/TFTP protocol
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
dhcp    - boot image via network using DHCP/TFTP protocol
echo    - echo args to console
erase   - erase FLASH memory
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
flinfo  - print FLASH memory information
go      - start application at address 'addr'
help    - print online help
imxtract- extract a part of a multi-image
itest   - return true/false on integer compare
loadb   - load binary file over serial line (kermit mode)
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
md      - memory display
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nand     - NAND sub-system
nboot   - boot from NAND device
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sleep   - delay execution for some time
tftpboot- boot image via network using TFTP protocol
usb     - USB sub-system
```



```
usbboot - boot from USB device
version - print monitor version
```

To get access to U-Boot console you have to press any key at boot:

```
RomBOOT
>AT91Boot-20081201

U-Boot 2009.01 (Mar 20 2009 - 13:43:24)

DRAM: 64 MB
NAND: 1024 MiB
In: serial
Out: serial
Err: serial
Net: macb0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
Hit any key to stop autoboot: 0
U-Boot>
```

Source code of U-Boot adapter to MMnet1001/1002 boards can be found on CD in archive /sources/u-boot-2009.01-MMnet1000.tar.gz. Configuration file at91sam9260ek was used and modified to work with modules.

3.5. Linux Kernel

Modules use standard 2.6.29.3 Linux kernel version, which has full support of AT91SAM9260/AT91SAM9G20 microprocessors. Sources adapted to MMnet1000 modules can be found on CD in archive: /sources/linux-2.6.29.3-MMnet1000.tar.gz. Kernel documentation can be found at <http://www.kernel.org/doc/> and in sources in Documentation directory.

Peripherals supported by kernel:

- Serial ports
- Ethernet
- USB Host
- USB Device
- MMC / SD
- I2C
- SPI
- SSC
- RTT (as RTC)
- Watchdog
- NAND Flash
- Compact Flash

3.6. The System

Linux system delivered with module is based on OpenWrt distribution (<http://openwrt.org/>).

Documentation and user forum can be found on above website.

In base system following commands are available:

```
[, [[, addgroup, adduser, adjtimex, ar, arp, arping, ash,
awk, basename, bbconfig, brctl, bunzip2, bzip2, cat,
catv, chat, chgrp, chmod, chown, chpst, chroot, chrt, chvt,
cksum, clear, comm, cp, crond, crontab, cut, date, dd, delgroup,
deluser, df, dhcprelay, diff, dirname, dmesg, dnsd, dpkg,
dpkg-deb, du, dumpleases, echo, egrep, env, envdir, envuidgid,
ether-wake, expr, fakeidentd, false, fetchmail, fgrep, find,
fold, free, ftpget, ftpput, fuser, getty, grep, gunzip,
gzip, halt, head, hexdump, hostid, hostname, httpd, hwclock,
id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd,
insmod, install, ip, ipaddr, ipcalc, iplink, iproute, iprule,
iptunnel, kill, killall, killall5, klogd, last, length,
less, ln, lock, logger, login, logname, logread, ls, lsmod,
lzmacat, makedevs, md5sum, mdev, mesg, microcom, mkdir,
mkfifo, mknod, mkswap, mktemp, modprobe, more, mount, mountpoint,
mv, nameif, nc, netmsg, netstat, nice, nmeter, nslookup,
openvt, passwd, pgrep, pidof, ping, ping6, pivot_root, pkill,
poweroff, printenv, printf, ps, pscan, pwd, rdate, readahead,
readlink, realpath, reboot, renice, reset, resize, rm, rmdir,
rmmmod, route, rpm, rtcwake, runlevel, runsv, runsvdir, rx,
sed, sendmail, seq, setconsole, setkeycodes, setsid, setuidgid,
sh, sleep, softlimit, sort, split, start-stop-daemon, stat,
strings, stty, su, sulogin, sv, svlogd, swapoff, swapon,
switch_root, sync, sysctl, syslogd, tail, tar, tcpsvd, tee,
telnet, telnetd, test, tftp, tftpd, time, top, touch, tr,
traceroute, true, tty, ttysize, udhcpd, udpsvd,
umount, uname, uniq, unlzma, unzip, uptime, usleep, vconfig,
vi, watch, watchdog, wc, wget, which, who, whoami, xargs,
yes, zcat, zcip
```

In addition, dropbear (SSH client/server), wireless tools and mtd-tools packages are installed:

```
dropbear
iwconfig, iwgetid, iwlist, wpriv, iwspy
flash_erase, flash_eraseall, flash_info, flash_lock,
flash_unlock, flashcp, nanddump, nandtest, nandwrite, ubiattach,
ubidetach, ubiformat, ubimkvol, ubirmvol, ubiupdatevol
```

OpenWrt has packages management system opkg. On our webserver can be found about 300 software packages (their are also on CD). List of packages with description is here: <http://www.propox.com/download/linux/at91/packages/Packages.gz>

To list available packages in module console:

```
opkg update
opkg list
```

To install selected module:

```
opkg update  
opkg install module_name
```

(opkg update can be executed only once after every system reboot).

More options:

```
opkg
```

4. Arrangement of system components in NAND Flash

NAND Flash memory is divided into five MTD partitions which are described below:

Device	Beg. Addr.	End Addr.	Size	Name
mtd0	0x00000000	0x0003ffff	0x00040000	bootstrap
mtd1	0x00040000	0x0007ffff	0x00040000	u-boot
mtd2	0x00080000	0x001fffff	0x00180000	u-boot environment
mtd3	0x00200000	0x007fffff	0x00600000	Kernel
mtd4	0x00800000	0x3fffffff	0x3f800000	filesystems

Partition mtd0 contain AT91Boot bootloader, mtd2 contain environment data for u-boot (placed in mtd1), mtd4 is intended to fileststems.

On mtd4 partition UBI layer is placed, which in turn is divided into two volumes:

- „rootfs” - 128MB size
- „storage” - about 840MB size (rest of memory)

On top of both volumes UBIFS file systems are created. Volume „rootfs” is mounted at kernel startup as / and contain whole system. Volume „storage” is mounted at OpenWrt startup in /mnt/storage directory and is intended for user data storage.

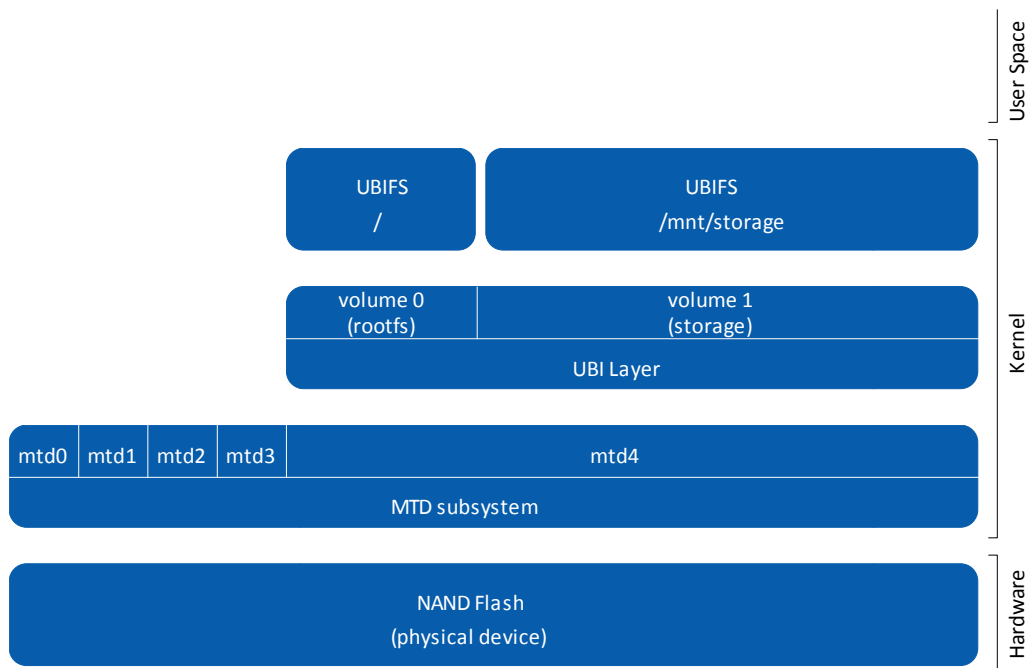


Figure 4 NAND Flash organisation.

Documentation of mtd subsystem, UBI layer, UBIFS file system and related to them mtd-utils can be found at: <http://www.linux-mtd.infradead.org/>

5. Software upgrade

To perform software upgrade you need PC computer with TFTP server installed, from which software will be downloaded. As TFTP server in Windows system may be used simple to configure Tftpd32, it can be found at: <http://tftpd32.jounin.net/> (it is included on CD). In Linux system, tftpd should be configured (description can be found in attachment to this document). TFTP server should be available at IP address 192.168.1.20.

5.1. Kernel upgrade

5.1.1. From Linux

Download new kernel image from TFTP server:

```
tftp -g -r uImage 192.168.1.20
```

Erase mtd partition with kernel:

```
flash_eraseall /dev/mtd3
```

Write new kernel to mtd partition:

```
nandwrite -p -m /dev/mtd3 uImage
```

5.1.2. From U-Boot

Download new kernel image from TFTP server:

```
tftp 0x22000000 192.168.1.20:uImage
```

Erase region in NAND Flash containing kernel:

```
nand erase 0x200000 0x800000
```

Write new kernel to NAND Flash:

```
nand write 0x22000000 0x200000 0x800000
```

Modules for new kernel should be copied to /lib/modules.

5.2. Installing system from scratch

WARNING: after this operations all data in NAND Flash memory will be lost!

PC should have installed SAM-BA (AT91-ISP) software from Atmel (on CD). Atmel officially deliver this software only for Windows system, there is also Linux equivalent, but it wasn't tested by us:

http://www.linux4sam.org/twiki/bin/view/Linux4SAM/SoftwareTools#SAM_BA_Linux_initiative

Local Ethernet Network, where module is connected, should have DHCP server.

1. Launch ROMBoot bootloader on module. To do this, you need to erase AT91Boot from NAND Flash.

To erase AT91Boot from Linux console:

```
flash_eraseall /dev/mtd0
```

To erase AT91Boot from U-Boot console:

```
nand erase 0x0 0x40000
```

If you don't have access to Linux nor U-Boot console, close SAMBA jumper on module (USE LED diode should lit) and two times switch on/off power. In DBGU terminal you should see "RomBOOT" (and no other messages after this). Remove SAMBA jumper.

2. Connect module's USB Device port to PC. In Device Manager new USB device „atm6124.Sys ATMEL AT91xxxx Test Board" should appear.
3. From flashing directory on CD launch "MMnet1000_prog.bat". After few seconds programming log should appear, you can close it (example of correct log file is in attachment to this document). AT91Boot and U-Boot bootloaders are now programmed to NAND Flash memory.
4. Make content of directory /flashing/tftp from CD available in TFTP server (at address 192.168.1.20).
5. After resetting module (it is necessary to cycle Power) U-Boot will launch and will download from TFTP server to SDRAM memory special version of Linux kernel prepared for programming system to Flash memory.
6. When Linux system start and console is available type:

```
./program
```

This will download system image and program it to NAND Flash. After programming Linux will reboot to newly programmed system.

6. Peripherals handling

6.1. Ethernet

Linux system prepared for module has full support for Ethernet interface. It is handled in standard Linux way.

```
root@MMnet:/# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 3A:1F:34:08:54:54
          inet addr:192.168.1.13  Bcast:192.168.1.255
Mask:255.255.255.0
          inet6 addr: fe80::381f:34ff:fe08:5454/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:1941 errors:0 dropped:0 overruns:0 frame:0
TX packets:55 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:218829 (213.7 KiB)  TX bytes:6655 (6.4 KiB)
Interrupt:21 Base address:0x4000
```

6.2. RS232

System prepared for module has by default configured three RS232 interfaces:

- DBGU as ttyS0 (system console, no modem signals)
- USART0 as ttyS1 (used modem signals: CTS, RTS, DTR, DSR, DCD, RI)
- USART1 as ttyS2 (used modem signals: CTS, RTS)

To add support for more interfaces or reconfigure existing it is necessary to modify file in kernel sources and recompile kernel. USART interfaces are registered in function `ek_map_io` in file `arch/arm/mach-at91/board-mmnet1000.c`.

Description and programming guide for Linux serial ports can be found in document „Serial Programming Guide for POSIX Operating Systems“:

<http://www.easysw.com/~mike/serial/serial.html>

6.3. USB Host

Onboard USB Host is fully supported by Linux kernel. Delivered system contain many kernel module supporting different USB devices like, storage devices, sound cards, Wi-Fi adapters, camera etc. Because module uses modern 2.6.29.3 kernel, most USB devices supported by standard Linux distributions, should also work with modules.

- Example of using USB hard disk:

```
modprobe sd_mod
modprobe vfat
mkdir /mnt/usbstorage/
mount /dev/sda1 /mnt/usbstorage/
```

(after connecting disc to USB bus in `/dev/` directory should automatically appear `sda1` device).

- Example of using USB Wi-Fi adapter with RT73 chipset:

Copy firmware file rt73.bin delivered with card to /lib/firmware directory on module:

```
cd /lib/firmware/
tftp -g -r rt73.bin 192.168.1.20
```

After connecting card to USB load rt73usb driver:

```
modprobe rt73usb
```

Configure wlan0 interface:

```
ifconfig wlan0 up
```

Configure Wi-Fi:

```
iwconfig wlan0 essid MY_NETWORK_NAME
```

Force dhcp client to obtain IP address from DHCP:

```
udhcpc -i wlan0
```

Available networks can be listed with iwlist:

```
root@MMnet:/# iwlist wlan0 scan
wlan0      Scan completed :
            Cell 01 - Address: 00:12:17:70:E0:A1
                        Channel:11
                        Frequency:2.462 GHz (Channel 11)
                        Quality=50/70  Signal level=-60 dBm
                        Encryption key:off
                        ESSID:"MY_NETWORK_NAME"
                        Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18 Mb/s
                                24 Mb/s; 36 Mb/s; 54 Mb/s
                        Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 48 Mb/s
                        Mode:Master
                        Extra:tsf=000000c8ef1c818f
                        Extra: Last beacon: 20ms ago
                        IE: Unknown: 0006507230703058
                        IE: Unknown: 010882848B962430486C
                        IE: Unknown: 03010B
                        IE: Unknown: 2A0107
                        IE: Unknown: 2F0107
                        IE: Unknown: 32040C121860
                        IE: Unknown: DD050010180104
```

6.4.USB Device

Onboard USB Device is fully supported by Linux kernel. Linux have „USB Gadget” framework which can work as different USB devices. Documentation can be found at: <http://www.linux-usb.org/gadget/>

6.5. MicroSD

To mount microSD card inserted into MMnet1002 module socket:

```
modprobe at91_mci
modprobe mmc_block
modprobe vfat
mkdir /mnt/mmc
mount /dev/mmcblk0p1 /mnt/mmc/
```

6.6. LED „USR”

Linux Kernel have LED subsystem, which documentation can be found in Documentation/LED-class.txt file in kernel sources.

In system delivered with module one LED device is registered. It has name usr and controls “USR” LED diode connected to PC15 port:

```
root@MMnet:/# ls /sys/class/leds/usr/
brightness device power subsystem trigger uevent
```

Every LED device is driven by „triggers”. Triggers available on module:

```
root@MMnet:/# cat /sys/class/leds/usr/trigger
none nand-disk mmc0 timer [heartbeat] default-on
```

By default „heartbeat” trigger is set, which blinks LED with frequency depended on system load.

To set another trigger, for example such that blinks LED during NAND Flash activity:

```
root@MMnet:/# echo nand-disk > /sys/class/leds/usr/trigger
```

To allow control of USR LED diode by user’s application, „none” trigger should be set:

```
root@MMnet:/# echo none > /sys/class/leds/usr/trigger
```

In this case LED is controlled by “brightness” file:

```
root@MMnet:/# echo 0 > /sys/class/leds/usr/brightness
root@MMnet:/# echo 1 > /sys/class/leds/usr/brightness
```

CD contain example software written in C, which blink USR LED diode with selected frequency.

More information about LED subsystem can be found in Linux kernel documentation.

6.7.GPIO

Linux kernel have support for controlling gpio ports from userspace. Documentation of gpio subsystem can be found in kernel sources in Documentation/gpio.txt file.

Access to gpio ports from userspace is possible through virtual filesystem /sys/:

```
root@MMnet:/# ls /sys/class/gpio/
export gpiochip32 gpiochip64 gpiochip96 unexport
```

To get access to selected gpio pin, it should be exported to userspace. In below examples PC7 pin (gpio103) will be used:

```
root@MMnet:/# echo 103 > /sys/class/gpio/export
```

After this operations, in /sys/class/gpio directory new subdirectory „gpio103” should appear:

```
root@MMnet:/# ls /sys/class/gpio/gpio103/
direction power      subsystem uevent      value
```

Gpio numbers for AT91SAM9260 gpio ports is presented in table:

Table 1 Gpio numbering.

port	gpio	port	gpio	port	gpio	port	gpio
PA0	gpio32	PB0	gpio64	PB16	gpio80	PC0	gpio96
PA1	gpio33	PB1	gpio65	PB17	gpio81	PC1	gpio97
PA2	gpio34	PB2	gpio66	PB18	gpio82	PC2	gpio98
PA3	gpio35	PB3	gpio67	PB19	gpio83	PC3	gpio99
PA4	gpio36	PB4	gpio68	PB20	gpio84	PC4	gpio100
PA5	gpio37	PB5	gpio69	PB21	gpio85	PC5	gpio101
PA6	gpio38	PB6	gpio70	PB22	gpio86	PC6	gpio102
PA8	gpio40	PB7	gpio71	PB23	gpio87	PC7	gpio103
PA9	gpio41	PB8	gpio72	PB24	gpio88	PC8	gpio104
PA23	gpio55	PB9	gpio73	PB25	gpio89	PC9	gpio105
PA24	gpio56	PB10	gpio74	PB26	gpio90	PC10	gpio106
PA30	gpio62	PB11	gpio75	PB27	gpio91	PC11	gpio107
PA31	gpio63	PB12	gpio76	PB28	gpio92	PC12	gpio108
		PB13	gpio77	PB29	gpio93	PC15	gpio111
		PB14	gpio78	PB30	gpio94		
		PB15	gpio79	PB31	gpio95		

To configure selected pin as input or output, „direction” file should be written with one of attributes: „in”, „out”, „high”, „low”. Attributes „high” i „low” configure pin as output with initial value. For example.:

```
root@MMnet:/# echo "low" > /sys/class/gpio/gpio103/direction
```

Direction file can be also read:

```
root@MMnet:/# cat /sys/class/gpio/gpio103/direction
out
```

To set or read pin state, „value” file should be used:

```
root@MMnet:/# echo 1 > /sys/class/gpio/gpio103/value
root@MMnet:/# cat /sys/class/gpio/gpio103/value
1
root@MMnet:/# echo 0 > /sys/class/gpio/gpio103/value
root@MMnet:/# cat /sys/class/gpio/gpio103/value
0
```

CD contain example program written in C, which blink LED diode connected to selected gpio port.

More information about gpio subsystem can be found in Linux kernel documentation.

Notice: Some gpio pins are reserved by other peripherals (RS232, MMC etc.) and cannot be used. Availability of gpio pins can be checked in table below.

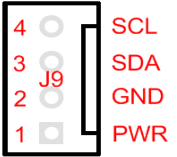
Table 2 MMnet1001/1002 IO ports

EVbmTm connector	Default function in Linux kernel	Gpio number	Alt. Function 1 [Alt. function 2]	Name/Port	J1		Name/Port	Alt. Function 1 [Alt. function 2]	Gpio number	Default function in Linux kernel	EVbmTm connector
A1			GND	GND	1	2	PC15	NWAIT [IRQ1]	gpio111	LED	B1
A1	GPIO	gpio64	SPI1_MISO [TIOA3]	PB0	3	4	PB1	SPI1_MOSI [TIOB3]	gpio65	GPIO	B2
A3	GPIO	gpio66	SPI1_SPCK [TIOA4]	PB2	5	6	PB3	SPI1_NPCS0 [TIOA5]	gpio67	GPIO	B3
A4	USART0	gpio68	TXD0	PB4	7	8	PB5	RXD0	gpio69	USART0	B4
A5	USART0	gpio70	TXD1 [TCLK1]	PB6	9	10	PB7	RXD1 [TCLK2]	gpio71	USART1	B5
A6	GPIO	gpio72	TXD2	PB8	11	12	PB9	RXD2	gpio73	GPIO	B6
A7	DBGU	gpio78	DRXD	PB14	13	14	PB15	DTXD	gpio79	DBGU	B7
A8	GPIO	gpio80	TK0 [TCLK3]	PB16	15	16	PB17	TF0 [TCLK4]	gpio81	GPIO	B8
A9	GPIO	gpio82	TD0 [TIOB4]	PB18	17	18	PB19	RD0 [TIOB5]	gpio83	GPIO	B9
A10			HDMA	HDMA	19	20	HDPB	HDPB			B10
A11			HDMA	HDMA	21	22	HDPB	HDPB			B11
A12			DDM	DDM	23	24	DDP	DDP			B12
A13			TDI	TDI	25	26	TDO	TDO			B13
A14			TMS	TMS	27	28	TCK	TCK			B14
A15			NTRST	NTRST	29	30	RTCK	RTCK			B15
A16			NRST	NRST	31	32	JTAGSEL	JTAGSEL			B16
A17			SHDN	SHDN	33	34	WKUP	WKUP			B17
A18	GPIO	gpio104	NCS4/CFCS0[RTS3]	PC8	35	36	PC9	NCS5/CFCS1[TIOB0]	gpio105	GPIO	B18
A19	-	gpio106	A25/CFRNW [CTS3]	PC10	37	38	PC11	NCS2 [SPI0_NPCS1]	gpio107	GPIO	B19
A20			GND	GND	39	40	PC12	IRQ0 [NCS7]	gpio108	GPIO	B20
					J2						
C1				+3.3V	1	2	GND				D1
C2	MMC	gpio32	SPI0_MISO[MICDB0]	PA0	3	4	PA1	SPI0_MOSI [MICDB]	gpio33	MMC	D2
C3	MMC	gpio34	SPI0_SPCK	PA2	5	6	PA3	SPI0_NPCS0[MICDB3]	gpio35	MMC	D3
C4	MMC	gpio36	RTS2 [MICDB2]	PA4	7	8	PA5	CTS2 [MICDB1]	gpio37	MMC	D4
C5	GPIO	gpio38	MCDA0	PA6	9	10	PA8	MCCK	gpio40	MMC	D5
C6	GPIO	gpio41	MCDA1	PA9	11	12	PA23	TWD [ETX2]	gpio55	GPIO/I2C	D6
C7	GPIO/I2C	gpio56	TWCK [ETX3]	PA24	13	14	PA30	SCK2 [RXD4]	gpio62	GPIO	D7
C8	GPIO	gpio63	SCK0 [TXD4]	PA31	15	16	PB10	TXD3 [ISI_D8]	gpio74	GPIO	D8
C9	GPIO	gpio75	RXD3 [ISI_D9]	PB11	17	18	PB12	TXD5 [ISI_D10]	gpio76	GPIO	D9
C10	GPIO	gpio77	RXD5 [ISI_D11]	PB13	19	20	PB20	RK0 [ISI_D0]	gpio84	GPIO	D10
C11	GPIO	gpio85	RF0 [ISI_D1]	PB21	21	22	PB22	DSR0 [ISI_D2]	gpio86	USART0	D11
C12	USART0	gpio87	DCD0 [ISI_D3]	PB23	23	24	PB24	DTR0 [ISI_D4]	gpio88	USART0	D12
C13	USART0	gpio89	RI0 [ISI_D5]	PB25	25	26	PB26	RTS0 [ISI_D6]	gpio90	USART0	D13
C14	USART0	gpio91	CTS0 [ISI_D7]	PB27	27	28	PB28	RTS1 [ISI_PCK]	gpio92	USART1	D14
C15	USART1	gpio93	CTS1 [ISI_VSYNC]	PB29	29	30	PB30	PCK0 [ISI_HSYNC]	gpio94	GPIO	D15
C16	GPIO	gpio95	PCK1 [ISI_MCK]	PB31	31	32	PC0	AD0 [SCK3]	gpio96	GPIO	D16
C17	GPIO	gpio97	AD1 [PCK0]	PC1	33	34	PC2	AD2 [PCK1]	gpio98	GPIO	D17
C18	GPIO	gpio99	AD3 [SPI1_NPCS3]	PC3	35	36	PC4	A23 [SPI1_NPCS2]	gpio100	-	D18
C19	-	gpio101	A24 [SPI1_NPCS1]	PC5	37	38	PC6	TIOB2 [CFCE1]	gpio102	GPIO	D19
C20	GPIO	gpio103	TIOB1 [CFCE2]	PC7	39	40	GND				D20

6.8.I2C

Linux kernel have I2C subsystem with drivers that supports many I2C devices. System delivered with module was configured to use following pins as I2C:

Table 3 I2C signals.

I2C signal	Port	Module connector	EVBMmTm socket pin	MMnet1002 connector
SDA	PA23	J2-12	D6	
SCL	PA24	J2-13	C7	

NOTICE: module's I2C bus work with 3.3V voltage levels and is **not** compatible with 5V signals !

To use I2C bus it is necessary to load following kernel module: i2c-gpio, i2c-core, i2c-algo-bit (last two will be loaded automatically after first). To use I2C from userspace it is also necessary to load i2c-dev module.

Applications that allow to use I2C bus can be found in i2c-tools package:

```
opkg update
opkg install i2c-tools
```

Available applications:

```
i2cdetect i2cdump i2cget i2cset
```

Example of use:

Loading kernel modules:

```
root@MMnet:/# modprobe i2c-gpio
i2c-gpio i2c-gpio: using pins 55 (SDA) and 56 (SCL)
```

```
root@MMnet:/# modprobe i2c_dev
i2c /dev entries driver
```

In /dev/ directory will be automatically created i2c device:

```
root@MMnet:/# ls /dev/ | grep i2c
i2c-0
```

Scanning bus with use of i2c-tools will show connected devices:

```

root@MMnet:/# i2cdetect -y 0
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  49  --  --  --  --  --  --
50: 50 51 52 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --

```

More information about I2C subsystem can be found in kernel sources in Documentation/i2c directory.

6.9. SPI

SPI subsystem documentation and sample software can be found in kernel sources in Documentation/spi directory. Kernel shipped with modules was not configured to use any of 2 available SPI buses. To configure this, it is necessary to modify file board-mmnet1000.c in arch/arm/mach-at91 dirctory in kernel sources and recompile kernel. It should be noted that SPI0 bus use the same gpio pins as MMC.

6.10. RTC

Hardware RTC clock is controlled with “hwclock” command. To configure it and set it to system time type in console:

```

root@MMnet:/# hwclock -w

```

System time can be set with “date” command, for example:

```

root@MMnet:/# date 2009.04.23-12:18:00
Thu Apr 23 12:18:00 UTC 2009

```

7. Software Development Kit and example applications

Software intended for module have to be compiled on PC computer with Linux system installed. CD delivered with module contain cross-compiler, it can be found in OpenWrt-SDK-at91-for-Linux-i686.tar.bz2 archive.

To make cross-compiler visible in system it should be added to path:

```
export PATH=$PATH:/path/to/OpenWrt-SDK-at91-for-Linux-  
i686/staging_dir/toolchain-arm_gcc4.1.2/bin
```

Where “/path/to/” should be replaced with actual path to directory unpacked from OpenWrt-SDK archive.

Software development kit is based on:

- gcc 4.1.2
- uClibc 0.9.29
- gdb 6.3

CD also contain example software. At the moment are available:

- led-blink – program that blink LED at selected port with selected frequency
- user-led-blink – program that blink USR LED with selected frequency
- spidev_test – program that show how to use SPI bus
- as programs that show how to use I2C bus i2c-tools and lm-sensors sources can be used: <http://www.lm-sensors.org/>
- programming guide and sample software for Linux serial ports can be found in document „Serial Programming Guide for POSIX Operating Systems”: <http://www.easysw.com/~mike/serial/serial.html>

To compile led-blink program:

```
arm-linux-uclibc-gcc led-blink.c -o led-blink
```

Generated ARM executable file should be copied to module. According to copy method, it may be necessary to set executable mode for file:

```
root@MMnet:/# tftp -g -r led-blink 192.168.1.20  
root@MMnet:/# chmod +x led-blink  
root@MMnet:/# ./led-blink 103 500  
Exporting gpio 103...OK  
Setting direction of gpio 103 to output...OK  
  
Press Ctrl+C to exit.  
  
Setting output value to 0  
Setting output value to 1  
Setting output value to 0  
Setting output value to 1
```

8. System compilation

8.1. AT91Boot compilation

AT91Boot bootloader sources, adapted for MMnet1001/1002 modules, are on CD in /sources/Bootstrap-v1.10-MMnet1000.tar.gz archive. To compile it type in board/MMnet1000/nandflash directory:

```
make CROSS_COMPILE=arm-linux-uclibc-
```

8.2. U-Boot-a compilation

U-Boot bootloader sources, adapted for MMnet1001/1002 modules, are on CD in /sources/u-boot-2009.01-MMnet1000.tar.gz archive. Configuration for board at91sam9260ek was used and modified to our modules needs. Configuration file is in include/configs/at91sam9260ek.h file. To recompile U-Boot unpack sources and type:

```
make CROSS_COMPILE=arm-linux-uclibc-
```

Generated u-boot.bin file can be used to flash module.

8.3. Linux kernel compilation

Linux kernel sources, adapted for MMnet1001/1002 modules, are on CD in /sources/linux-2.6.29.3-MMnet1000.tar.gz archive. After unpacking it should be configured for MMnet1000 boards:

```
make ARCH=arm CROSS_COMPILE=arm-linux-uclibc- mmnet1000_defconfig
```

To configure kernel:

```
make ARCH=arm CROSS_COMPILE=arm-linux-uclibc- menuconfig
```

To compile kernel:

```
make ARCH=arm CROSS_COMPILE=arm-linux-uclibc- uImage
```

After compiling in arch/arm/boot directory ulmage file should appear. It can be used to flash module.

To compile kernel modules:

```
make ARCH=arm CROSS_COMPILE=arm-linux-uclibc- modules
```

To install kernel modules:

```
make ARCH=arm CROSS_COMPILE=arm-linux-uclibc-  
INSTALL_MOD_PATH=/path/to/my/rootfs modules_install
```

Where /path/to/my/rootfs is a path where kernel module will be copied. They should be copied to module or used to create new filesystem image.

File specific to MMnet1000 modules and containing boars initialization code is in arch/arm/mach-at91/board-mmnet1000.c file.

8.4. OpenWrt compilation

OpenWrt system delivered with module was compiled from SVN revision 13340. Sources are included on CD.

Because OpenWrt build system can't generate UBI images, tar.gz should be selected as output format. UBI image should be prepared manually from this archive.

More information can be found in OpenWrt documentation:

<http://kamikaze.openwrt.org/docs/openwrt.html#x1-390002>

8.5. UBI and UBIFS images preparation

Two directories are needed : ./storage (empty directory for „storage” volume) and ./rootfs containing files for root filesystem. PC computer should also have mtd-utils packane installed.

First create UBIFS images:

```
mkfs.ubifs -r ./storage -m 2048 -e 258048 -c 4096 -o MMnet1000-  
storage.ubifs
```

```
mkfs.ubifs -r ./rootfs -m 2048 -e 258048 -c 4096 -o MMnet1000-OpenWrt-  
rootfs.ubifs
```

Then UBI image containing volumes with UBIFS filesystems:

```
ubinize -o MMnet1000-OpenWrt.ubi -m 2048 -p 256KiB ubinize.cfg
```

(where ubinize.cfg is configuration file, it can be found on CD and in attachment in this document). Created MMnet1000-OpenWrt.ubi file contain UBI image which can be programmed to Flash memory.

Documentation of mtd subsystem, UBI layer, UBIFS file system and associated with them mtd-utils can be found at: <http://www.linux-mtd.infradead.org/>

9. FAQ

9.1. During system boot messages „Bad eraseblock 1843 at 0x1ccc0000” appear. Is this normal?

It is normal. NAND Flash memories can have bad blocks even when they are new. Bad blocks count will increase during memory lifetime but will not exceeds 100 (it is guaranteed by memory manufacturer). UBU layer and UBIFS filesystem used in module guarantee protection from data loss.

9.2. microSD do not work with MMnet1002

Check if your module have R44 resistor mounted, if it has, unsolder it. Some of first modules was shipped with this resistor mounted.

10. Useful links

<http://kernel.org/>
<http://www.denx.de/wiki/U-Boot/WebHome>
<http://openwrt.org/>
<http://www.linux4sam.org>
<http://www.at91.com>
<http://www.network-theory.co.uk/gcc/intro/>
<http://www.linux-mtd.infradead.org/>
<http://www.linux-usb.org/>
<http://www.linux-usb.org/gadget/>
<http://elinux.org>

11. License, warranty and technical support

Most of software delivered with module has GNU GPL license or similar. Details can be found in software sources or in its documentation.

Software delivered with module is free, open source software. It is delivered “as is”, with hope that will be useful, but without any warranty. Propox company does not take any responsibility for this software.

We can provide only basic technical support in matters closely related to our MMnet1001/1002 modules. We do not provide support for general Linux matters nor for delivered by us software which is not written by us.

Technical support is provided only by e-mail. Please send questions to: support@propox.com

12. Attachments

12.1. System boot log

```
RomBOOT
>AT91Boot-20081201

U-Boot 2009.01 (Mar 20 2009 - 13:43:24)

DRAM: 64 MB
NAND: 1024 MiB
In: serial
Out: serial
Err: serial
Net: macb0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
Hit any key to stop autoboot: 0

NAND read: device 0 offset 0x200000, size 0x200000
2097152 bytes read: OK
## Booting kernel from Legacy Image at 22000000 ...
Image Name: Linux-2.6.28.8
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1469276 Bytes = 1.4 MB
Load Address: 20008000
Entry Point: 20008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing
Linux.....
..... done, booting the kernel.
Linux version 2.6.28.8 (user@MMnet1000DevEnv) (gcc version 4.1.2) #1
PREEMPT Fri Mar 20 11:07:45 CET 2009
CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ), cr=00053177
CPU: VIVT data cache, VIVT instruction cache
Machine: Propox MMnet1000
Memory policy: ECC disabled, Data cache writeback
Clocks: CPU 198 MHz, master 99 MHz, main 18.432 MHz
Built 1 zonelists in Zone order, mobility grouping on. Total pages:
16256
Kernel command line: mem=64M console=ttyS0,115200 ubi.mtd=4
root=ubi0:rootfs rootfstype=ubifs init=/etc/preinit
AT91: 96 gpio irqs in 3 banks
PID hash table entries: 256 (order: 8, 1024 bytes)
Console: colour dummy device 80x30
console [ttyS0] enabled
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
```

```

Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 61852KB available (2712K code, 197K data, 100K init)
Calibrating delay loop... 98.91 BogoMIPS (lpj=494592)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
net_namespace: 636 bytes
NET: Registered protocol family 16
AT91: Power Management
AT91: Starting after user reset
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
NET: Registered protocol family 23
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NET: Registered protocol family 1
NetWinder Floating Point Emulator V0.97 (double precision)
JFFS2 version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
msgmni has been set to 120
alg: No test for stdrng (krng)
io scheduler noop registered
io scheduler deadline registered (default)
atmel_uart.0: ttyS0 at MMIO 0xfefff200 (irq = 1) is a ATMEL_SERIAL
atmel_uart.1: ttyS1 at MMIO 0xffffb0000 (irq = 6) is a ATMEL_SERIAL
atmel_uart.2: ttyS2 at MMIO 0xffffb4000 (irq = 7) is a ATMEL_SERIAL
loop: module loaded
MACB_mii_bus: probed
eth0: Atmel MACB at 0xfffc4000 irq 21 (3a:1f:34:08:54:54)
eth0: attached PHY driver [Davicom DM9161A]
(mii_bus:phy_addr=ffffffff:00, irq=-1)
NAND device: Manufacturer ID: 0x2c, Chip ID: 0xd3 (Micron NAND 1GiB 3,3V
8-bit)
Scanning device for bad blocks
Bad eraseblock 1843 at 0x1ccc0000
Bad eraseblock 3063 at 0x2fdc0000
Bad eraseblock 3129 at 0x30e40000
Bad eraseblock 3202 at 0x32080000
Creating 5 MTD partitions on "atmel_nand":
0x00000000-0x00040000 : "bootstrap"
0x00040000-0x00080000 : "u-boot"
0x00080000-0x00200000 : "u-boot environment"
0x00200000-0x00800000 : "kernel"
0x00800000-0x40000000 : "filesystems"
UBI: attaching mtd4 to ubi0
UBI: physical eraseblock size: 262144 bytes (256 KiB)
UBI: logical eraseblock size: 258048 bytes
UBI: smallest flash I/O unit: 2048
UBI: VID header offset: 2048 (aligned 2048)

```

```

UBI: data offset:                4096
UBI: attached mtd4 to ubi0
UBI: MTD device name:            "filesystems"
UBI: MTD device size:            1016 MiB
UBI: number of good PEBs:        4060
UBI: number of bad PEBs:         4
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    256
UBI: number of internal volumes: 1
UBI: number of user volumes:     2
UBI: available PEBs:             0
UBI: total number of reserved PEBs: 4060
UBI: number of PEBs reserved for bad PEB handling: 40
UBI: max/mean erase counter: 28/5
UBI: background thread "ubi_bgt0d" started, PID 266
usbmon: debugfs is not available
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
at91_ohci at91_ohci: AT91 OHCI
at91_ohci at91_ohci: new USB bus registered, assigned bus number 1
at91_ohci at91_ohci: irq 20, io mem 0x00500000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
usb usb1: New USB device found, idVendor=1d6b, idProduct=0001
usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb1: Product: AT91 OHCI
usb usb1: Manufacturer: Linux 2.6.28.8 ohci_hcd
usb usb1: SerialNumber: at91
usbcore: registered new interface driver libusual
mice: PS/2 mouse device common for all mice
rtc-at91sam9 at91_rtt.0: rtc core: registered at91_rtt as rtc0
cpuidle: using governor ladder
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
Registered led device: usr
Netfilter messages via NETLINK v0.30.
TCP cubic registered
NET: Registered protocol family 17
rtc-at91sam9 at91_rtt.0: setting system clock to 1970-02-22 20:30:09 UTC
(4566609)
UBIFS: recovery needed
UBIFS: recovery completed
UBIFS: mounted UBI device 0, volume 0, name "rootfs"
UBIFS: file system size: 131862528 bytes (128772 KiB, 125 MiB, 511
LEBs)
UBIFS: journal size: 9420800 bytes (9200 KiB, 8 MiB, 37 LEBs)
UBIFS: media format: 4 (latest is 4)
UBIFS: default compressor: LZO
UBIFS: reserved for root: 0 bytes (0 KiB)
VFS: Mounted root (ubifs filesystem).
Freeing init memory: 100K
Warning: unable to open an initial console.
- preinit -

```

```

Press CTRL-C for failsafe
- init -

Please press Enter to activate this console. PPP generic driver version
2.4.2
UBIFS: recovery needed
UBIFS: recovery completed
UBIFS: mounted UBI device 0, volume 1, name "storage"
UBIFS: file system size:      899297280 bytes (878220 KiB, 857 MiB, 3485
LEBs)
UBIFS: journal size:         9420800 bytes (9200 KiB, 8 MiB, 37 LEBs)
UBIFS: media format:         4 (latest is 4)
UBIFS: default compressor: LZO
UBIFS: reserved for root:    0 bytes (0 KiB)
eth0: link up (100/Full)
NET: Registered protocol family 10

BusyBox v1.11.3 (2008-12-03 11:05:39 CET) built-in shell (ash)
Enter 'help' for a list of built-in commands.

      _____
      |         |.-----|.-----|.-----|.         |         |.-----|.         | | | | | |
      |  -      ||  _  | -__|         ||         ||         ||         ||         |
      |_____|  ||  _  |_____|_____|_____|_____|_____|_____|_____|_____|_____|
              |__| W I R E L E S S   F R E E D O M
KAMIKAZE (bleeding edge, r13340) -----

Propox MMnet1000
www.propox.com
-----

root@MMnet:/#

```

12.2. U-Boot environment variables for system programming

```

ethact=macb0
ethaddr=3a:1f:34:08:54:54
serverip=192.168.1.20
ipaddr=192.168.1.55
baudrate=115200
bootdelay=1
bootargs=mem=64M console=ttyS0,115200 root=/dev/ram0 init=/etc/preinit
bootcmd=tftp 0x22000000 uImage-prog; bootm 0x22000000
stdin=serial
stdout=serial
stderr=serial

```

12.3. U-Boot environment variables

```

ethact=macb0
ethaddr=3a:1f:34:08:54:54
serverip=192.168.1.20
ipaddr=192.168.1.55
baudrate=115200

```

```

bootdelay=1
bootargs=mem=64M console=ttyS0,115200 ubi.mtd=4 root=ubi0:rootfs
rootfstype=ubifs init=/etc/preinit
bootcmd=nand read 0x22000000 0x00200000 0x00200000; bootm 0x22000000
stdin=serial
stdout=serial
stderr=serial

```

12.4. Script for software upgrade

```

#!/bin/sh

udhcpc -f -q -A 3

echo "Getting files from TFTP server"

tftp -g -r uImage 192.168.1.20
tftp -g -r MMnet1000-OpenWrt.ubiimg 192.168.1.20
tftp -g -r uboot-env.bin 192.168.1.20

echo "Writing rootfilesystem"

ubiformat /dev/mtd4 -f MMnet1000-OpenWrt.ubiimg

echo "Writing Linux kernel"

flash_eraseall /dev/mtd3
nandwrite -p /dev/mtd3 uImage

echo "Writing new U-Boot env. file"

flash_eraseall /dev/mtd2
nandwrite -p /dev/mtd2 uboot-env.bin

echo "Rebooting to new system"

reboot

```

12.5. Ubinize configuration file

```

[rootfs-volume]
mode=ubi
image=MMnet1000-OpenWrt-rootfs.ubifsimg
vol_id=0
vol_size=128MiB
vol_type=dynamic
vol_name=rootfs

[storage-volume]
mode=ubi
image=MMnet1000-storage.ubifsimg
vol_id=1
vol_size=800MiB
vol_type=dynamic
vol_name=storage

```



```
vol_flags=autoresize
```

12.6. SAM-BA programming log

```
-I- Waiting ...
connection : \usb\ARM0
board : AT91SAM9260-EK
target(handle) : 17976840
read chip ID : 0x00000010 at addr: 0xFFFFFEE40
read chip ID : 0x019803A2 at addr: 0xFFFFF240
-I- Found processor : AT91SAM9260 (0x019803A0)
-I- Loading applet isp-extram-at91sam9260.bin at address 0x200000
-I- Memory Size : 0x4000000 bytes
-I- Buffer address : 0x2007C4
-I- Buffer size: 0x0 bytes
-I- Applet initialization done
-I- External RAM initialized
script file : MMnet1000_prog.tcl
-I- === Initialize the NAND access ===
-I- NANDFLASH::Init (trace level : 3)
-I- Loading applet isp-nandflash-at91sam9260.bin at address 0x20000000
-I- Memory Size : 0x40000000 bytes
-I- Buffer address : 0x200047E4
-I- Buffer size: 0x40000 bytes
-I- Applet initialization done
-I- === Load the bootstrap: nandflash_at91sam9-ek in the first sector
===
GENERIC::SendFile AT91Boot_nandflash_MMnet1000.bin at address 0x0
-I- File size : 0xFF4 byte(s)
-I-      Writing: 0xFF4 bytes at 0x0 (buffer addr : 0x200047E4)
-I-      0xFF4 bytes written by applet
-I- === Load the u-boot in the next sectors ===
-I- Send File u-boot_nandflash.bin at address 0x00040000
GENERIC::SendFile u-boot_nandflash.bin at address 0x40000
-I- File size : 0x29F30 byte(s)
-I-      Writing: 0x29F30 bytes at 0x40000 (buffer addr : 0x200047E4)
-I-      0x29F30 bytes written by applet
-I- === Load the u-boot environment variables ===
-I- Send File uboot-env-prog.bin at address 0x00080000
GENERIC::SendFile uboot-env-prog.bin at address 0x80000
-I- File size : 0x40000 byte(s)
-I-      Writing: 0x40000 bytes at 0x80000 (buffer addr : 0x200047E4)
-I-      0x40000 bytes written by applet
```

12.7. How to configure TFTP server i Ubuntu Linux

Install tftpd and related packaged:

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

Create /etc/xinetd.d/tftp file with content:

```
service tftp
{
    protocol          = udp
```

```
port          = 69
socket_type   = dgram
wait          = yes
user          = nobody
server        = /usr/sbin/in.tftpd
server_args   = /tftpboot
disable       = no
}
```

Create /tftpboot file which will contain files available in ftp server:

```
sudo mkdir /tftpboot
sudo chmod -R 777 /tftpboot
sudo chown -R nobody /tftpboot
```

Start tftpd:

```
sudo /etc/init.d/xinetd start
```