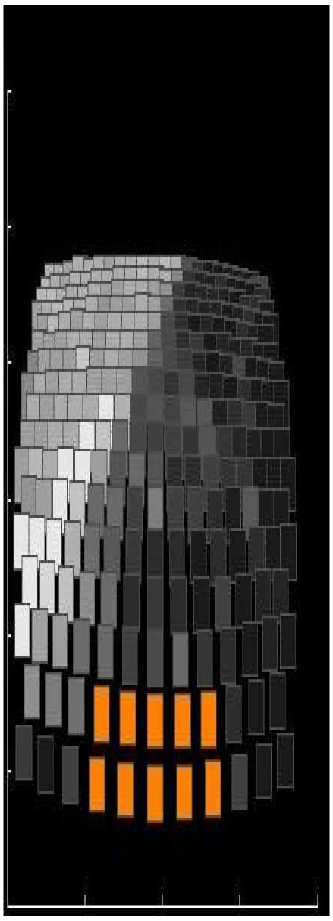


MBS270 V2

EMBEDDED VISION COMPUTER

MBS270 V2 Software Manual

Edition August 2009

Revision History

Date	Revision	Description
September 2008	-001	Initial release
January 2009	-002	Updated toolchain installation (Chapter 2) Corrected gdb setup Updated standalone/NFS operation (Chapter 3) Added SDcard basic operation. Added precision about GPIO, PWM and timer access. Updated GPIO driver interface. Minor number is set dynamically for all robotics oriented devices. Updated ADC driver interface. Added RTC section. Updated image acquisition chapter, support for all camera modules.
August 2009	-003	Added installation details for Ubuntu base station Modules location change Added crop feature video acquisition driver

Contents

I. INTRODUCTION

I.1 MBS270 V2 general presentation

MBS270 V2 is an embedded computer dedicated to control and computer vision. It is based on Intel/Marvell Technology PXA270 processor and offers multiple interfaces including digital camera.

The PXA270 processor is an integrated system-on-a-chip microprocessor for high-performance, low-power, portable, handheld and handset devices. It incorporates the Intel XScale technology with on-the-fly voltage and frequency scaling and sophisticated power management to provide industry-leading MIPs/mW performance. The PXA27x processor complies with the ARM Architecture V5TE instruction set (excluding floating point instructions) and follows the ARM programmer's model. The PXA27x processor also supports Intel Wireless MMX integer instructions in applications such as those that accelerate audio and video processing. It offers many peripherals such as 3 UARTs, 3 SSP, I2C, USB host and client controllers, external memory devices interfaces, 4 PWM and digital camera interface.

MBS270 V2 is a single-board computer bringing the richness and performance of PXA270 processor to robot and embedded control designers:

- A ready-to-run digital camera interface enables real-time image processing and vision control. Color and grey-level images can be acquired in various formats. Images are transferred to memory without CPU intervention via DMA.
- Many communication ports are available to interface with other systems: UART, SSP, I2C, USB host, Ethernet, SDIO.
- Additional peripherals are available: PWM, ADC, GPIO and timers, LCD with touchscreen, audio input and output.
- microSD card and USB flash drive can be added to extend storage capacity.
- All interfaces are easily accessible through convenient and reliable connectors.

MBS270 V2 has been designed for optimal performance in embedded systems:

- light and small: 46 grams[1], 50mm x 72mm board size
- low power: 1 Watt while executing 25fps image processing
- single power supply with on-board high efficiency converters.

MBS270-1 is easy to use thanks to available software running with it:

- Linux operating system
- Video For Linux 2 (V4L2) image acquisition driver
- Intel Integrated Performance Primitives (IPP) Library including many WMMX optimized routines for signal and image processing.
- MOBISENSE SYSTEMS additional drivers
- gdb and Eclipse
- Open source software from OpenEmbedded, Angstrom distribution…

[1] MBS270 V2 board with Colibri module only.

I.2 MBS270-1 features

I.2.1 Intel/Marvell PXA270 embedded computer

- Intel/Marvell PXA270 processor running at 520MHz
- 64 Mbytes of 32 bits SDRAM
- 32 Mbytes of 32 bits Flash
- Linux 2.6.26 operating system

I.2.2 Camera interface

- Standard 10 bits digital interface with Pixel clock, Line and Frame Valid signals operating up to 26MHz
- Master clock up to 26MHz
- FFC connection: high speed signals interleaved with ground preserve signal integrity and allow sensor to be placed far away from the port.
- I2C signals
- 3V3 and 5V power supplies

I.2.3 Communication ports

- Ethernet 10/100 Mbit
- 2 x USB host 1.1 ports
- 3 x asynchronous serial ports with all signals: 2 x RS232, 1 x TTL
- 2 x synchronous serial ports SSP/SPI/Microwire/PSP compatible
- I2C
- SDIO

I.2.4 Multimedia

- 20 bits audio: stereo headphone output, stereo line input and microphone input
- up to 18 bits LCD port
- XY touch screen inputs

I.2.5 Additional peripherals

- 4 x PWM outputs
- 4 x 10bits ADC inputs
- Up to 31 GPIOs[1], 4 driving LEDs, 4 driven by micro switch
- 1 event count input
- 1 timer event output
- Battery saved real time clock RTC
- Power supervision signals

[1] Depending of multiplexed pins use.

I.3 Block Diagram

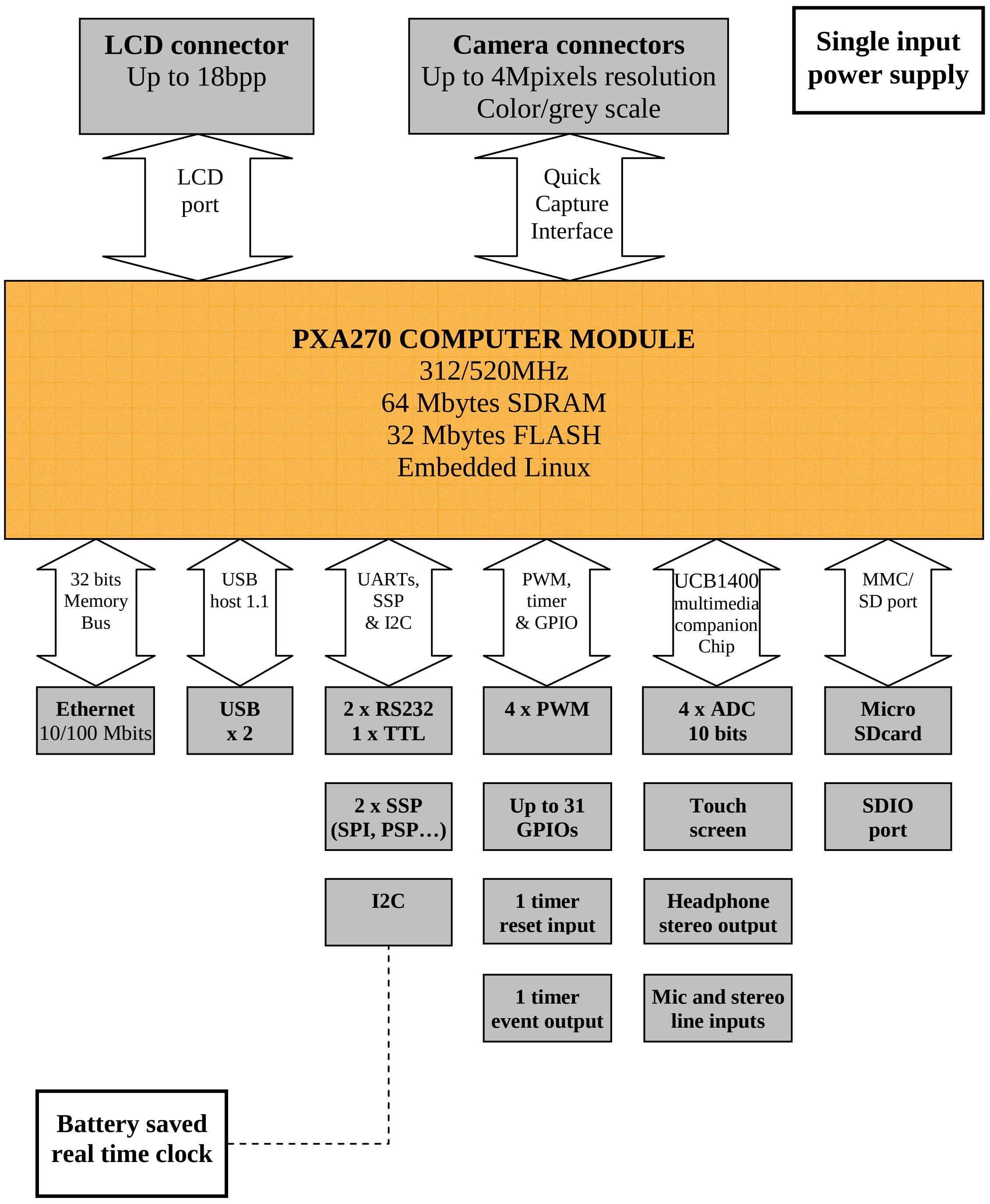


Figure 1: Block diagram

I.4 About this manual

This manual presents how to develop software applications for MBS270. In this manual, it is assumed that the reader has a basic knowledge of Linux operating system and C language programming.
Chapter II Development toolchain installation explains how to install the development toolchain on your computer.
Chapter III MBS270 operation and configuration contains basic operation guidelines and various procedures to adapt the system to your specific needs.
Image acquisition is detailed in Chapter IV.
Access to robotics special features is explained in Chapter V.

II. DEVELOPMENT TOOLCHAIN INSTALLATION

This chapter is inspired from Detlef Vollmann document “getting-started.txt” included in Colibri BSP 2.2.

II.1 Requirements

For developing with MBS270, you need a computer workstation that meets the following requirements:

Hardware:

- Pentium compatible CPU
- 256 MB RAM
- 1 RS232 serial port
- 1 Ethernet port (2 ports recommended)
- 500 MB free disk space

Software:

- Linux operating system (Kernel 2.6.x recommended)
- DHCP server
- TFTP server
- NFS server
- minicom or other serial terminal emulation program
- GCC 3.x

Instructions in this chapter should essentially work for all Linux distributions. There are some minor differences between distributions for the network setup. If the instructions given in this chapter don't work for your development workstation, you should ask your system administrator how to accomplish the respective tasks on your specific system setup.

MBS270 can run:

- as a standalone computer with kernel and file system in flash. This is the typical way to use it as an embedded computer. Only TFTP server is needed to transfer programs or data files between the workstation and MBS270 (If you plan to use MBS270 only that way, you can skip sections II.3.3 DHCP Server, II.3.5 NFS Server and II.4 Target software installation).
- via NFS mount with kernel downloaded with TFTP at boot time. This is a convenient way to use it during application development.

This chapter explains how to set up these tools on your workstation, MBS270 set-up for standalone or NFS operation is detailed in sections III.2.5 and III.2.6.

II.2 Board Support Package (BSP) download

MBS270 BSP can be downloaded at:
http://www.mobisensesystems.com/fics/xscale_BSPs/yyyy_mm
In the following sections, we suppose the whole BSP tree structure is stored on a CDROM (which is recommended) in the BSP_yymm directory and the CDROM mount point is /media/cdrom.

II.3 Host preparations

Note: Some of the following steps require root privileges.

For development, several software pieces are required on the development workstation (or in the development environment):

- one or two GCC toolchains for cross-compiling programs for the PXA270,
- a DHCP server to tell MBS270 on boot time an IP address, the server to boot from, the file to load for booting, and the server and path for the root filesystem,
- a TFTP server to supply MBS270 with the boot image,
- an NFS server to mount the root file system from.

The following sections explain how to setup this required software on the development workstation.

It is useful to have everything in one place, e.g. /work/mbs270 and then define an environment variable for it:

```
 $ mkdir -p /work/mbs270
 $ export MBS270=/work/mbs270
```

You should add a line such as this to your shell startup file (.profile, .bashrc, .cshrc, ...).

II.3.1 GCC Toolchains

Two GCC toolchains are included in the BSP:

- The GNU EABI[1] toolchain is the most recent toolchain, with significant performance enhancement, especially for floating point computation. MBS270 Linux kernel is built with this toolchain and it is recommended to use it to build your own applications.
  See http://wiki.debian.org/ArmEabiPort for a general presentation.
- The older toolchain is required to link with non-EABI libraries such as Intel IPP. MBS270 Linux kernel is able to execute non EABI applications.
  Applications built with this toolchain can not link dynamically with system shared libraries so –static option shall be specified to the linker. See IPP examples for Makefile syntax.

GNU EABI toolchain installation:

You must be root to install the toolchain.

```
 $ cd /
 $ tar xvjf /media/cdrom/BSP_yymm/system/toolchains/arm-linux-
gnueabi.tar.bz2
```

This creates a full toolchain tree in /usr/local/arm/oe.

Now, you need to include the /usr/local/arm/oe/bin directory into your PATH:

```
 $ PATH=$PATH:/usr/local/arm/oe/bin
```

Put a statement like that into your shell startup file, e.g. .bashrc or .profile.

To test your toolchain installation, you can simply compile an empty file:

```
 $ touch xyz.c
 $ arm-linux-gnueabi-gcc --version -c xyz.c
```

This should give you something like:

```
 arm-linux-gnueabi-gcc (GCC) 4.1.2
```

[1] Embedded Application Binary Interface

...

Non EABI toolchain installation:
You must be root to install the toolchain.

```
 $ cd /
 $ tar xvjf /media/cdrom/BSP_yymm/system/toolchains/arm-linux.tar.bz2
```

This creates a full toolchain tree in /usr/local/arm-linux.
Now, you need to include the /usr/local/arm-linux/bin directory into your PATH:

```
 $ PATH=$PATH:/usr/local/arm-linux/bin
```

Put a statement like that into your shell startup file, e.g. .bashrc or .profile.

To test your toolchain installation, you can simply compile an empty file:

```
 $ touch xyz.c
 $ arm-linux-gcc --version -c xyz.c
```

This should give you something like:

```
 arm-linux-gcc (GCC) 3.3.2
 ...
```

II.3.2 General Network Setup

For a number of reasons, the best network setup for development is a separate physical connection between your workstation and MBS270. For this connection, these instructions assume the interface eth1 on your workstation and the subnet 192.168.1.0/24.
You should check that this subnet doesn't conflict with the normal network setup (eth0) of your workstation.
If you don't have a physically separate network connection, but connect MBS270 to a shared network, you need to check the following procedures with your network administrator!
In this case, you probably should not setup your on own DHCP server, but add MBS270 to an existing DHCP server. And the setup of a TFTP Server and an NFS server might be a security risk.

To setup the Ethernet interface for the private connection to the MBS270, you can simply issue a respective ifconfig command:

```
 $ ifconfig eth1 192.168.1.1
```

But you should add the interface configuration to the startup procedure.
For a Debian or Ubuntu system, you need to add an entry to /etc/network/interfaces:

```
  auto eth1
  iface eth1 inet static
        address 192.168.1.1
        netmask 255.255.255.0
        network 192.168.1.0
        broadcast 192.168.1.255
```

For a RedHat system, you need to add a file /etc/sysconfig/network-scripts/ifcfg-eth1:

```
DEVICE=eth1
USERCTL=no
ONBOOT=yes
BOOTPROTO=none
BROADCAST=192.168.1.255
NETWORK=192.168.1.0
NETMASK=255.255.255.0
IPADDR=192.168.1.1
```

For a SuSE system, you should either use Yast, or edit /etc/rc.config manually, so that it contains the following entries:

```
NETCONFIG="_0 _1"
IPADDR_1="192.168.1.1"
NETDEV_1="eth1"
IFCONFIG_1="192.168.1.1 broadcast 192.168.1.0 netmask 255.255.255.0"
```

You can test the configuration with

```
 $ ifconfig eth1
```

which should give you the correct IP address (192.168.1.1) and 'UP'.

II.3.3 DHCP Server

If the DHCP server dhcpd is not already installed on your workstation, use the package system for your distribution to install it (dhcp3-common and dhcp3-server for Debian and Ubuntu).

The configuration for dhcpd is generally controlled by two parts:

- the command line at invocation that tells the server on which network interface to operate,
- the configuration file that contains the information for the clients.

The first item is very important: you typically don't want to let your workstation answer any other DHCP requests than those from MBS270.

To specify that the DHCP server should only answer requests on the interface eth1, that information must be given on the command line for the dhcpd call, and that is usually in the init script /etc/init.d/dhcp or /etc/init.d/dhcpd.
But on most distributions, this script takes the interface from another configuration file:
For Debian or Ubuntu, in /etc/default/dhcp3-server you should set the variable INTERFACES to "eth1".
For RedHat, in /etc/sysconfig/dhcpd you should set the variable DHCPDARGS=eth1.
For SuSE, it's the same file, but the variable is DHCPD_INTERFACE="eth1".

The information configuration file for the DHCP server is in most distributions /etc/dhcpd.conf (/etc/dhcp3/dhcpd.conf for Debian and Ubuntu).
You should set it to something like the following:

```
# DHCP configuration file for MBS270 development

subnet 192.168.1.0 netmask 255.255.255.0 {
# --- default gateway
#   option routers              192.168.1.1;
    option subnet-mask          255.255.255.0;
    option broadcast-address    192.168.1.255;

    option domain-name          "my_domain";
    option domain-name-servers  192.168.1.1;

    option ip-forwarding        off;

    default-lease-time          86400;
    max-lease-time              86400;
    host mbs270 {
        hardware ethernet       00:14:2D:00:01:DD;
        fixed-address           192.168.1.95;
        option host-name        "mbs270";
        next-server             192.168.1.1;
        filename                "/work/mbs270/boot/uImage";
        option root-path        "192.168.1.1:/work/mbs270/rootfs/nfs";
    }
}
```

Some descriptions for the entries in the configuration file:

'option routers':	MBS270 default gateway;
'hardware ethernet':	MBS270 default MAC address;
'option host-name':	MBS270 hostname;
'next-server':	the IP address of the TFTP server;
'filename':	the name of the Linux image to be booted;
'option root-path':	the IP address of the NFS server together with the path of the target's root file system on the server.

The DHCP server requires an existing leases file, so make sure that it exists:

```
 $ touch /var/lib/dhcp/dhcpd.leases
```

The DHCP server is started through the init script:

```
$ /etc/init.d/dhcpd start
```

or

```
$ /etc/init.d/dhcp3-server start
```

or

```
 $ /etc/init.d/dhcp start
```

After a reboot, it should start automatically, if the link in the respective /etc/rcX.d directory exists.

II.3.4 TFTP Server

If the TFTP server (usually tftpd or in.tftpd) is not already installed on your workstation, use the package system for your distribution to install it.
For Ubuntu, xinetd, tftpd and tftp packages must be installed. An example is given at
http://www.davidsudjiman.info/2006/03/27/installing-and-setting-tftpd-in-ubuntu/

The TFTP server is usually started through the inetd server, so you need to activate it in /etc/inetd.conf. This file probably already has an entry like the following:

```
tftp  dgram  udp  wait  nobody  /usr/sbin/tcpd /usr/sbin/in.tftpd
/work/mbs270/boot
```

You should make sure that the line is not commented out and that the last field is correct: it restricts access of the TFTP server to the given directory.
If an additional '-s' is given, you should probably remove it or make sure that the 'filename' entry in the DHCP configuration is relative to the directory specified after the '-s'.

To declare an additional directory for your application programs for example, just add it at the end of this line:

```
tftp  dgram  udp  wait  nobody  /usr/sbin/tcpd /usr/sbin/in.tftpd
/work/mbs270/boot /home/robert/bin
```

If you made changes to /etc/inetd.conf, you need to restart it:

```
$ /etc/init.d/inetd restart
```

or

```
$ /etc/init.d/networking restart
```

or

```
 $ killall -HUP inetd
```

You should make sure that the user 'nobody' has execute and read permissions on the directory where the boot image resides.

II.3.5 NFS Server

The NFS server usually sits in the Linux kernel; if you have a custom kernel you need to make sure that it is enabled. If the user space parts of the NFS server are not already installed on your workstation, use the package system for your distribution to install it.

The file /etc/exports defines which filesystems are accessible for which clients via NFS. It should contain a line like the following:

```
/work/mbs270/rootfs/nfs    192.168.1.0/255.255.255.0(rw,no_root_squash)
```

If that line exists, you should restart your NFS server with the respective init script (depending on your distribution):
For Debian or Ubuntu:

```
 $ /etc/init.d/nfs-common restart
 $ /etc/init.d/nfs-kernel-server restart
```

or for SuSE

```
 $ /etc/init.d/nfsserver restart
```

or for RedHat

```
 $ /etc/init.d/nfs restart
```

II.4 Target software installation

II.4.1 Target file system

To boot via NFS, MBS270 needs the root file system at the location specified by the 'root-path' option in DHCP. Unpack MBS270 file system as follows with root privileges:

```
$ cd $MBS270
$ mkdir rootfs
$ cd rootfs
$ tar xvjf /media/cdrom/BSP_yymm/system/rootfs/nfs_rootfs.tar.bz2
```

II.5 Additional tools

II.5.1 gdb, the GNU Project Debugger

It is possible to debug applications with gdb via the gdb server. This works with a serial or a TCP/IP interface. To get into details about gdb, use:
```
$ info /usr/local/arm/oe/share/info/gdb.info
```

The following procedure is based on the source example in:
`/media/cdrom/BSP_yymm/app/tests/gdb.`

Build binaries for debug:
As indicated in the Makefile, two binaries are necessary:
- one without debug information on the target,
- one with debug information on the workstation, specify `–g` flag for that.

Launch debugger:
We focus on debug via TCP/IP since it is available on MBS270.
Launch the gdb server on MBS270 as follows:
```
root@mbs270:~$ cd tests
root@mbs270:~/tests$ gdbserver :port_ppp test_gdb
Process test_gdb created; pid = xxx
Listening on port port_ppp
```

Then connect to the server on the workstation side:
```
$ cd /media/cdrom/BSP_yymm/app/tests/gdb
$ arm-linux-gnueabi-gdb test_gdb
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-linux --target=arm-linux-gnueabi"...
(gdb) target remote mbs270_ip_address:port_ppp
Remote debugging using mbs270_ip_address:port_ppp
```

At this point, the following message is displayed on the target side:
```
Remote debugging from host workstation_ip_address
```

It is now possible to start debugging the application code, not the shared libraries as indicated by the warnings:
```
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x400008a0 in ?? ()
 (gdb) b main
Breakpoint 1 at 0x8400: file test_gdb.c, line 13.
 (gdb) c
Continuing.
```

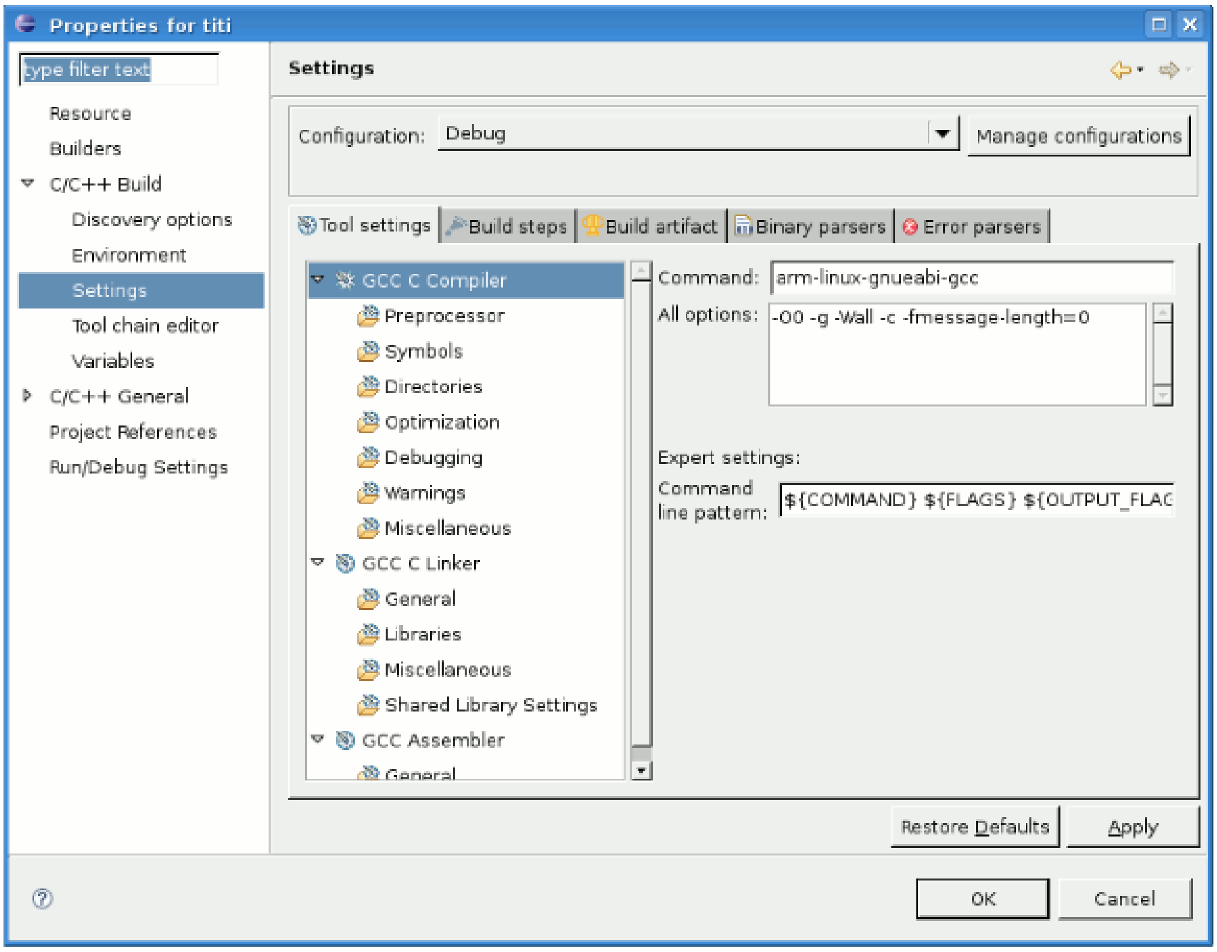
```
warning: .dynamic section for "/lib/libc.so.6" is not at the expected
address (wrong library or version mismatch?)
Error while mapping shared library sections:
/lib/ld-linux.so.3: No such file or directory.

Breakpoint 1, main (argc=1, argv=0xbe9b4e54) at test_gdb.c:13
13        printf("\n\n   ***   test_gdb   ***\n\n");
(gdb)
```

Use the help command to know about gdb commands or read the documentation.

II.5.2 Eclipse IDE

Eclipse Integrated Development Platform makes development of applications much easier.
We encourage visiting http://www.eclipse.org for information about this IDE.
As a first step, Eclipse CDT (C/C++ Development Tooling) shall be installed. Note that Eclipse is running with Java so you need a Java virtual machine on your workstation (sun-java6-jre and sun-java6-bin packages on Debian for example).

A few settings are necessary for cross compilations: specify `arm-linux-gnueabi-gcc` as compiler and linker commands in the project settings as shown in figures below.



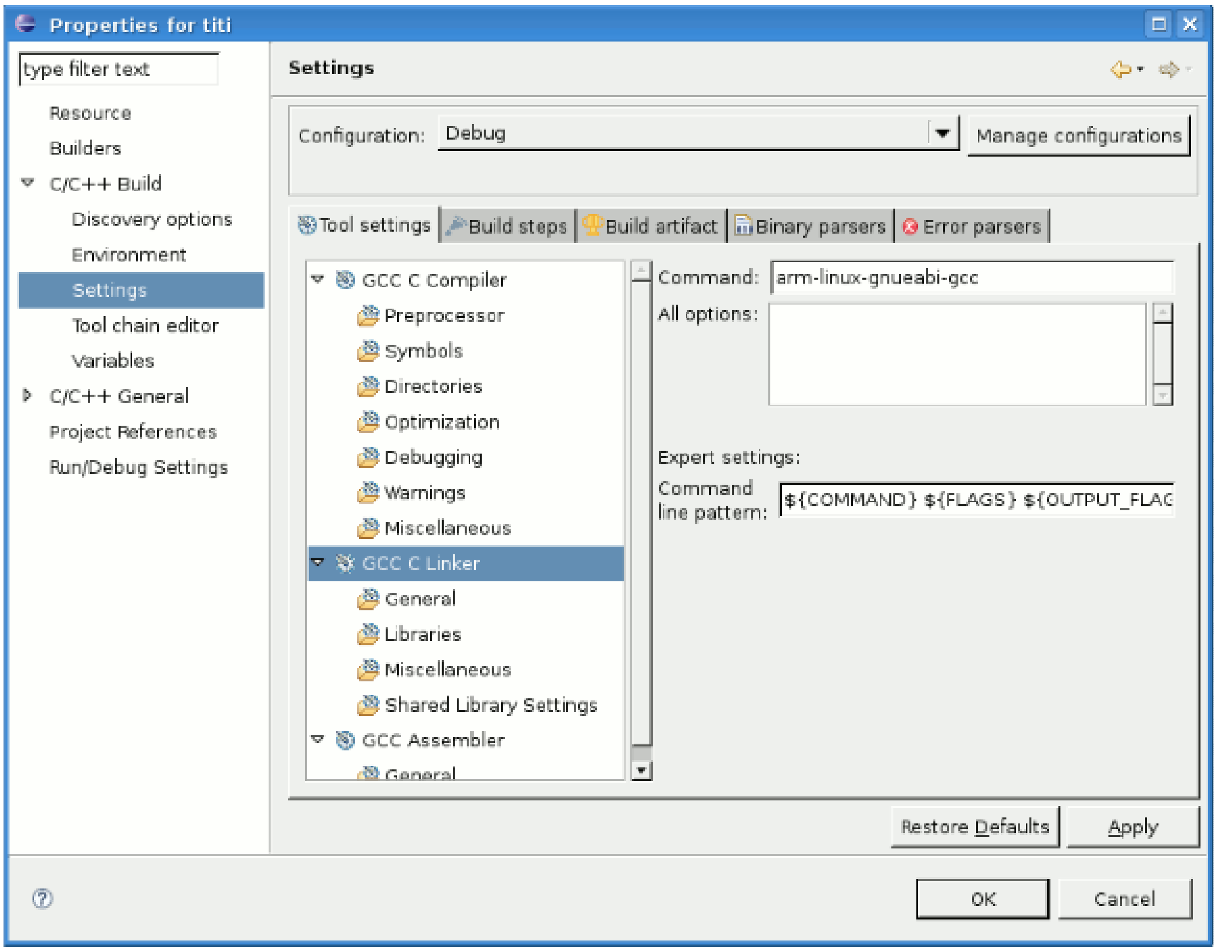
Configuration: Debug

Command: arm-linux-gnueabi-gcc

All options: -O0 -g -Wall -c -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG



Configuration: Debug

Command: arm-linux-gnueabi-gcc

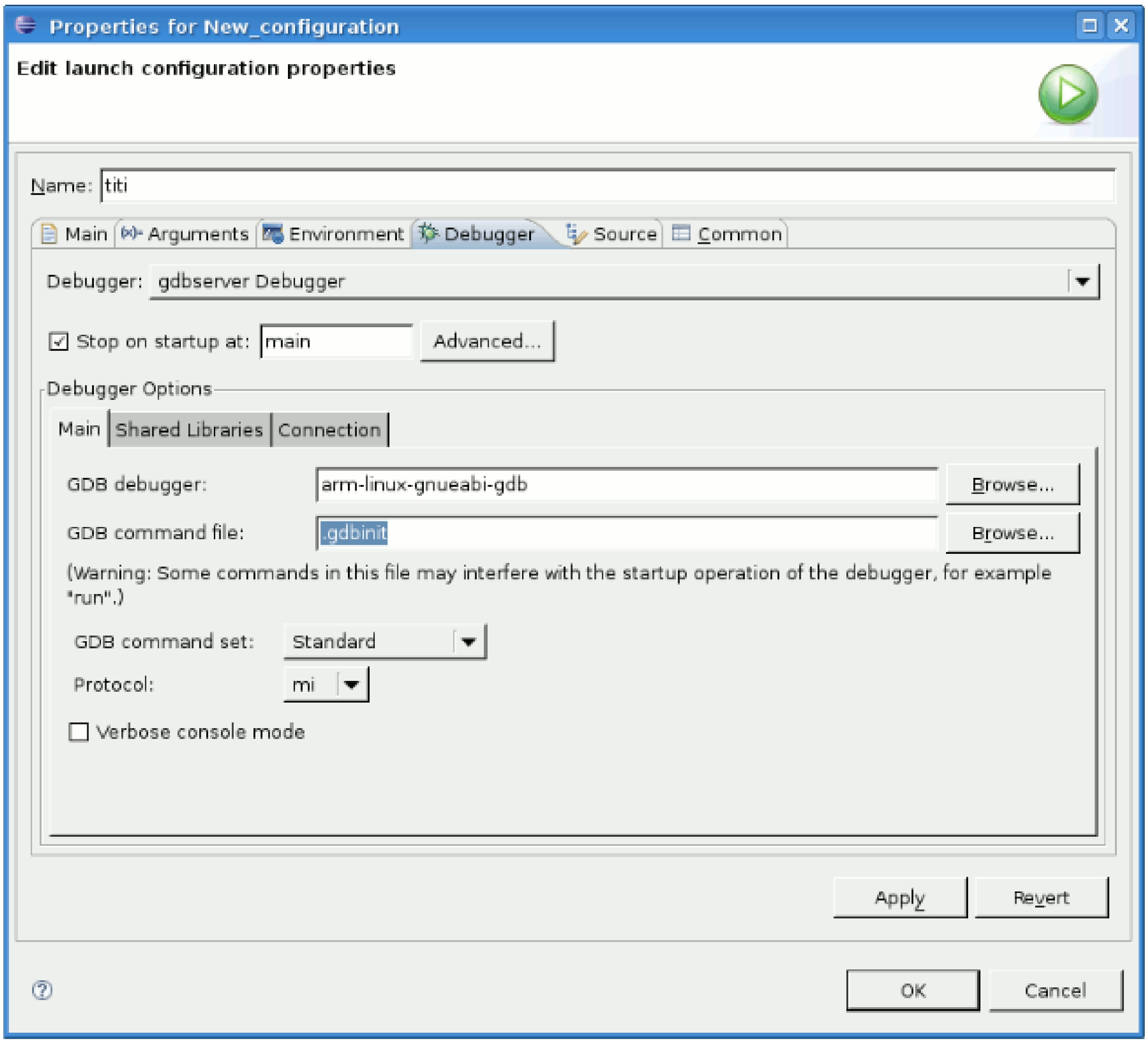
All options:

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG

II.5.3 gdb in Eclipse

Embedding gdb in Eclipse gives a graphical interface to the debugger.
You need to create a new configuration in the Run/Debug Settings a your project. In the Debugger tab of this configuration, enter the following data:


Properties for New_configuration
Edit launch configuration properties

Name: titi

Main | Arguments | Environment | Debugger | Source | Common

Debugger: gdbserver Debugger

☑ Stop on startup at: main   Advanced...

Debugger Options

Main | Shared Libraries | Connection

GDB debugger: arm-linux-gnueabi-gdb   Browse...

GDB command file: .gdbinit   Browse...

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

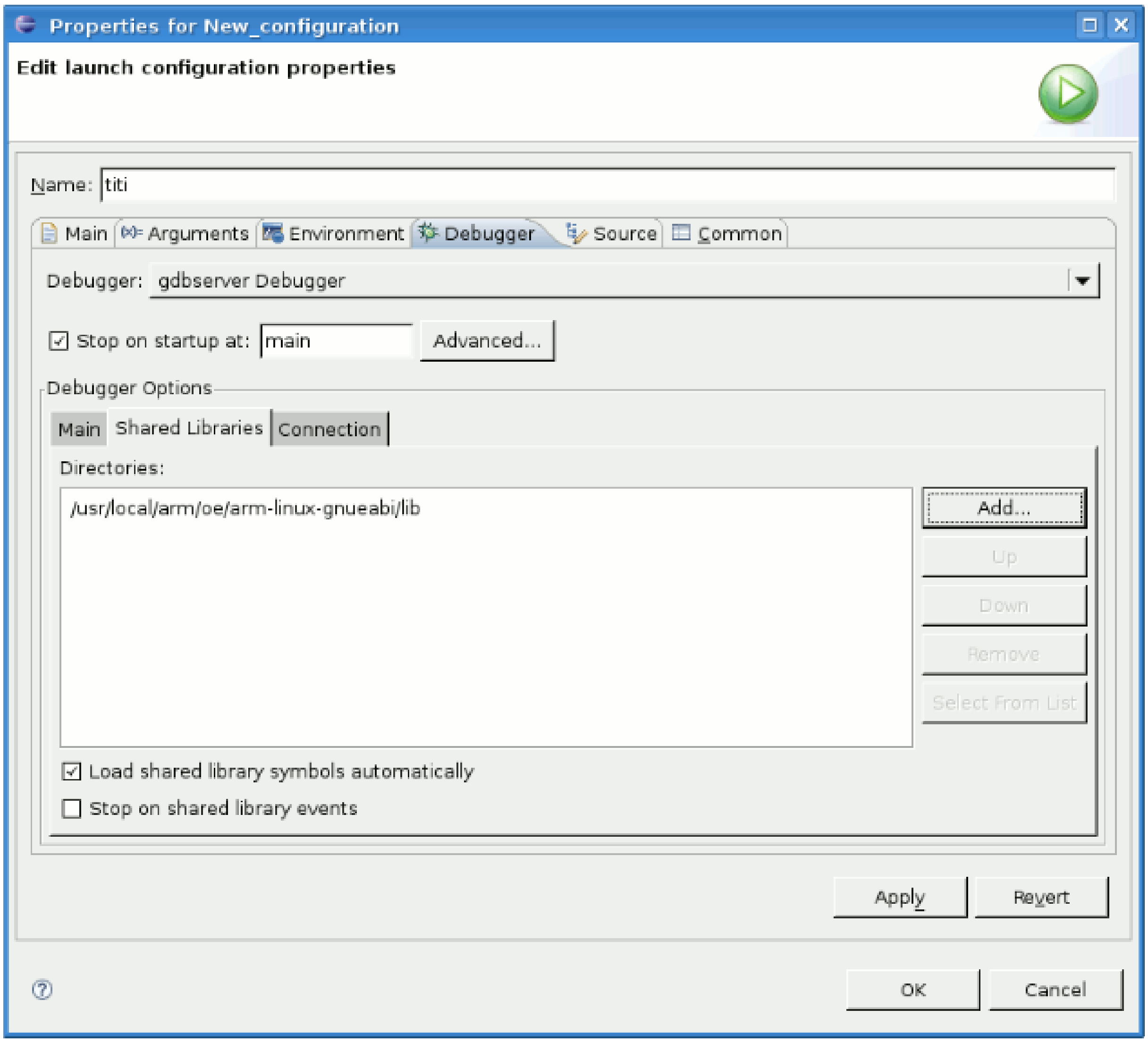
GDB command set: Standard

Protocol: mi

☐ Verbose console mode

Apply   Revert

OK   Cancel



Properties for New_configuration

Edit launch configuration properties

Name: titi

Main | Arguments | Environment | Debugger | Source | Common

Debugger: gdbserver Debugger

☑ Stop on startup at: main

Debugger Options

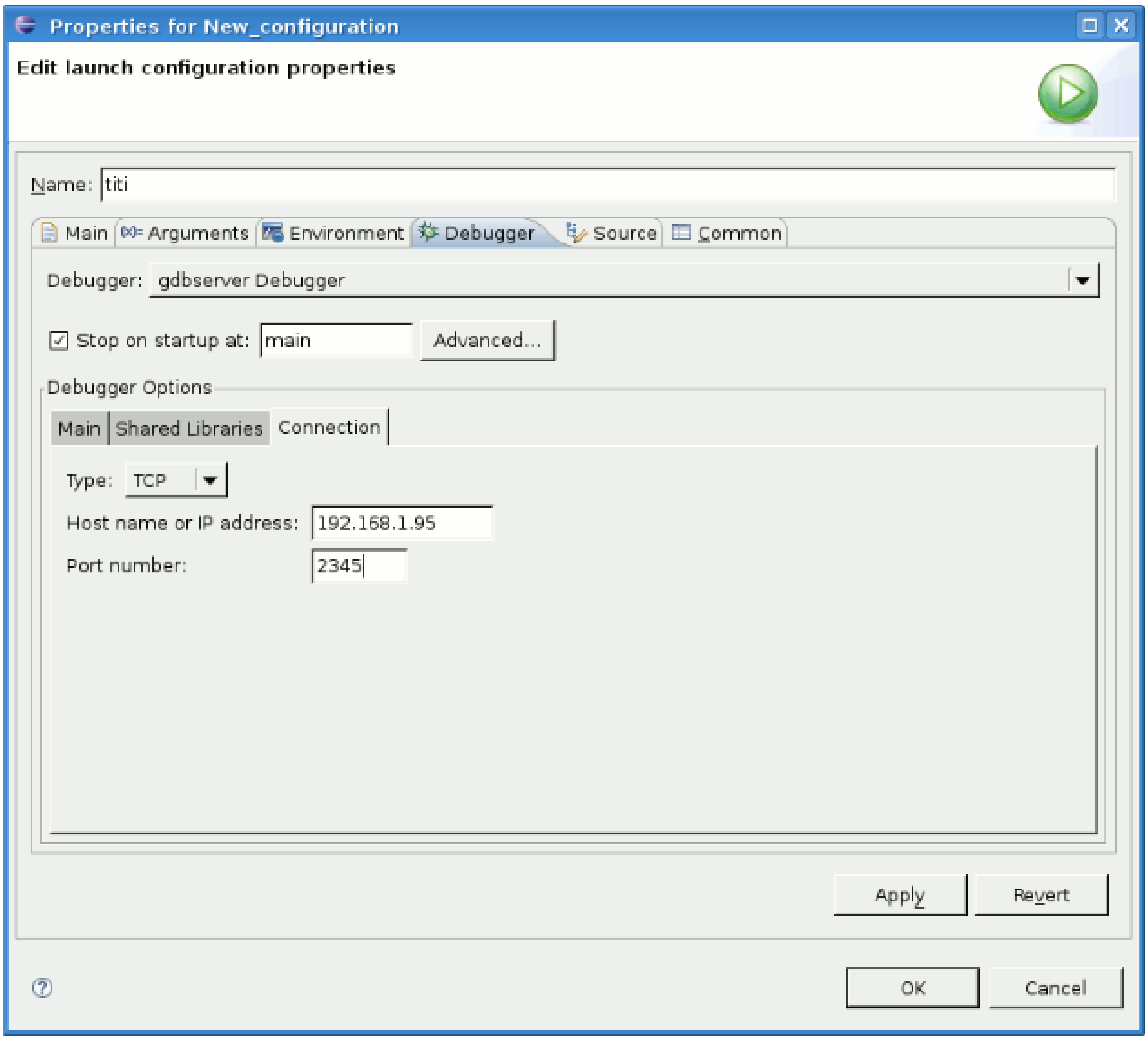
Main | Shared Libraries | Connection

Directories:

/usr/local/arm/oe/arm-linux-gnueabi/lib

☑ Load shared library symbols automatically

☐ Stop on shared library events



Name: titi

Debugger: gdbserver Debugger

☑ Stop on startup at: main

Debugger Options

Type: TCP

Host name or IP address: 192.168.1.95

Port number: 2345

III. MBS270 OPERATION AND CONFIGURATION

III.1 Standalone operation

By default, MBS270 is set for standalone operation, which means it is ready to run and starts running as soon as it is powered.

III.1.1 Flash memory partitions

Standalone operation is possible thanks to preinstalled system in flash. The flash memory is divided in four partitions handled as block devices by Linux. The partitions are:

- /dev/mtdblock0: contains u-boot bootloader, the initial program started after reset that will load Linux.
- /dev/mtdblock1: contains u-boot configuration parameters.
- /dev/mtdblock2: contains the Linux kernel.
- /dev/mtdblock3: contains the file system in jffs2 format.

III.1.2 Console output

Console output is bound to /dev/ttyS0 which is FFUART on PXA270. You can see the boot progress with a serial terminal program such as Minicom by connecting MBS270 FFUART to the serial port of your workstation. Since the workstation serial port works with RS232 levels, use RS232 level adapter included in the wiring kit. Console output for U-boot and Linux is set to /dev/ttyS0 with common setting 9600-8-N-1:

- Speed = 9600 bauds,
- No parity,
- 1 stop bit.

III.1.3 Shell

Once Linux boot sequence is finished, a shell is launched on /dev/ttyS0.
Default user is root, password is empty.
Once logged, you can execute system commands just like on your Linux workstation. Note that system commands are emulated with BusyBox and not all of them are emulated because of system disk space. Enter

```
root@mbs270:~# busybox
```

to see a list of current supported commands.
By default, Dropbear SSH server is launched at the end of the boot sequence. You can log in via SSH which gives a much faster interface than serial console.

III.1.4 File transfer

Files can be transferred between MBS270 and workstation with the TFTP protocol. The TFTP client runs on MBS270, the server on the workstation. Directories accessible by the client are defined in /etc/ inetd.conf as mentioned in II.3.4.
To transfer a file from the workstation to MBS270, enter:

```
root@mbs270:~# tftp –g –l mbs270_file –r /home/robert/bin/workstation_file
workstation_ip_address
```

- Option –g means get from server,
- Option –l specifies local file name, on client side,

- Option –r specifies remote file name, on server side; the path must be declared in /etc/ inetd.conf.

To transfer a file from MBS270 to the workstation, enter:

```
root@mbs270:~# tftp –p –l mbs270_file –r /home/robert/bin/workstation_file
file workstation_ip_address
```

- Option –p means put to server.

III.1.5 Flash file system considerations

Most Linux file system is read only and Linux operation induces write to files only in /var and /tmp. To avoid excessive writes to on board flash, /var is a partition mounted in virtual memory and /tmp is a symbolic link to /var/tmp as you can see with the following commands:

```
root@mbs270:~# mount
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
/dev/root on /dev/.static/dev type jffs2 (rw)
tmpfs on /dev type tmpfs (rw)
devpts on /dev/pts type devpts (rw)
tmpfs on /var type tmpfs (rw)
root@mbs270:~# ls -l /tmp
lrwxrwxrwx    1 root     0              8 Jan 16  2008 /tmp -> /var/tmp
root@mbs270:~#
```

The result of the *mount* command depends of course of the content of /etc/fstab file.

Important Note: applications running on MBS270 should also avoid frequent writes to on board flash. If an application needs to save large data files, it is strongly recommended to store these files on a microSD card or an USB mass storage device.

III.2 NFS operation

NFS operation makes file transfers transparent which is very convenient during application development. NFS operation is similar to standalone except that the whole system is stored in directories on the workstation. The Linux kernel is stored in a file at a specific location that is downloaded via TFTP at boot-up by u-boot and the file system is stored on the workstation hard drive (see Chapter II for NFS set up details). Of course, MBS270 must be connected via Ethernet to the workstation for NFS to work.

III.2.1 Flash memory partitions

You can access to flash memory partitions with the mount command. For example:

```
root@mbs270:~# mount -t jffs2 /dev/mtdblock3 /mnt/local
```

gives access to the onboard flash file system in /mnt/local directory. That way it is possible to modify the flash file system for specific operation.
It is not recommended to mount /dev/mtdblock0 through /dev/mtdblock2 that contain only binary data.

III.2.2 Console output

See III.1.2.

III.2.3 Shell

See III.1.3.

III.2.4 File transfer

MBS270 file system is a part of the workstation file system. File transfer can be executed with local commands.

III.2.5 Switching from standalone to NFS operation

The way Linux is loaded at boot time is selected by u-boot bootloader. U-boot behavior can be configured with environment variables (see u-boot overview III.3).
Two environment variables must be modified to boot via NFS, bootcmd and bootargs:
```
u-boot$ setenv bootcmd "dhcp && bootm"
u-boot$ setenv bootargs root=/dev/nfs ip=:::::eth0: console=ttyS0,9600n8
```
while others must be kept at their default values as shown in III.3.2.
The first command tells u-boot to download the Linux kernel via TFTP after DHCP request. The second command defines the arguments for the Linux kernel.
Save the new configuration in flash and reset:
```
u-boot$ saveenv
Saving Environment to Flash...
Un-Protected 1 sectors
Erasing Flash...
Erasing sector  1 ...  done
Erased 1 sectors
Writing to Flash...\done
Protected 1 sectors
u-boot$ reset
resetting …
```
MBS270 will now boot with the NFS file system.

III.2.6 Switching from NFS to standalone operation

bootcmd and bootargs environment variables must be set back to their default values as shown in III.3.2:
```
u-boot$ setenv bootcmd "bootm 80000"
u-boot$ setenv bootargs "root=/dev/mtdblock3 rootfstype=jffs2
console=ttyS0,9600n8"
```
The first command tells u-boot to load the Linux kernel at address 0x80000. The second command defines the arguments for the Linux kernel.
Save the new configuration in flash and reset:
```
u-boot$ saveenv
Saving Environment to Flash...
Un-Protected 1 sectors
Erasing Flash...
Erasing sector  1 ...  done
Erased 1 sectors
Writing to Flash...\done
Protected 1 sectors
u-boot$ reset
resetting …
```
MBS270 will now boot with the Flash file system.

III.3 U-boot

U-boot is MBS270 bootloader, the program called after board reset to launch the Linux kernel. U-boot behavior is configured with environment variables. To modify these variables, autoboot must be stopped to get control over u-boot.

III.3.1 Getting control over u-boot

By default, the serial console enables communication with u-boot. The console output (/dev/ttyS0) must be connected to a serial terminal (see III.1.2 for details).
Autoboot in u-boot can be stopped by pressing any key or by entering a specific character string from the terminal connected to ttyS0.
The string must be entered within a number of seconds specified by the bootdelay variable. Its default value is 1 second.
By default, the string is "ty" which is easy to enter within 1 second delay.
To stop autoboot process, just press alternatively t and y keys after power on until the u-boot prompt is displayed:

```
U-Boot 1.1.2 (Dec  1 2007 - 11:58:58)

U-Boot code: A3F80000 -> A3F98F38  BSS: -> A3F9D670
RAM Configuration:
Bank #0: a0000000  0 kB
Bank #1: 00000000  0 kB
Bank #2: 00000000  0 kB
Bank #3: 00000000  0 kB
Flash: 32 MB
In:    serial
Out:   serial
Err:   serial
Missing Colibri config block
u-boot$
```

The string can be modified with the *bootstopkey* variable definition:

```
setenv bootstopkey my_string
saveenv
```

At the next boot, my_string must be entered to stop autoboot.
For various options about u-boot, please consult doc/README.autoboot in u-boot source archive.

III.3.2 Default environment

The printenv command shows the current environment variables and their values. Below are the default values for MBS270:

```
u-boot$ printenv
bootdelay=1
stdin=serial
stdout=serial
stderr=serial
baudrate=9600
ethaddr=00:14:2D:00:01:DD
bootcmd=bootm 80000
bootargs=root=/dev/mtdblock3 rootfstype=jffs2 console=ttyS0,9600n8
pxaspeed=0x00001eb0
```

Note that the order the variables appear may differ on your MBS270. They are grouped here by function.

General:

bootdelay is the number of seconds u-boot waits for keystroke or “ty” input before starting to load Linux.

Console:

stdin, *stdout* and *stderr* define how standard streams are connected, *serial* means with the serial console.

baudrate is the speed of the serial connection of the console.

Linux kernel TFTP transfer with DHCP request. Values of these variables must be set in accordance with contents of file mentioned in chapter II:

ethaddr is the MAC address of the MBS270.

Boot commands (see III.2.5 and III.2.6 for alternate values):

bootcmd defines the way to boot the Linux kernel.

bootargs are the arguments passed to the Linux kernel.

III.3.3 Modify the environment

Variables are modified with the setenv command:

```
setenv variable_name variable_value
```

The new environment is saved with `saveenv` command. Changes are applied by resetting MBS270 with the `reset` command.

III.3.4 Silent boot

Silent boot significantly shortens the boot sequence because of the slow rate of the serial console. Silent boot is enabled by setting a non null value to `silent` variable:

```
setenv silent 1
saveenv
```

III.4 Alternate use of serial ports 0 and 1

This section details how default serial port usage can be modified.

By default, serial port 0 is the system console. During the boot sequence, system messages are output on this port. It is also via this port that you can configure u-boot bootloader.
At the end of the boot sequence, a getty is started on serial port 0 and optionally on port 1.

III.4.1 Alternate usage of serial port 1

If only one serial port is needed, this one should be used because the software modification is very simple. To disable getty launch after boot, just comment out the line starting with S1 in /etc/inittab by inserted a # at the beginning:

```
#S1:2345:respawn:/sbin/getty ttyS1 9600
```

To enable it again, uncomment the line by removing the #.

III.4.2 Alternate usage of serial port 0

This modification is more complex because console output and getty must be disabled.
Console output is disabled at u-boot level. u-boot disables Linux console by itself.
To disable u-boot console output, start the system and stop the autoboot process by entering the appropriate key sequence (see III.3.1).
Then define the silent variable (the value is not important, only the definition is necessary):

```
setenv silent 1
saveenv
reset
```

The system will reboot without displaying messages on ttyS0. At the end of the boot sequence, getty is started and the login prompt is displayed.

To disable getty launch after boot, just comment out the line starting with S0 in /etc/inittab by inserted a # at the beginning:

```
#S0:2345:respawn:/sbin/getty ttyS0 9600
```

When the system is restarted, nothing at all is output on serial port 0 and no process will consume inputs. This way, the serial port is available for any use.

Note: after reset, u-boot waits bootdelay seconds for a specific string input to stop the autoboot process. This specific string must be chosen so that the device connected to serial port 0 can not send a similar string otherwise autoboot will never start. If a specific string can not be defined, the Rx pin of serial port 0 must be disconnected from the device during the bootdelay.

To enable serial port 0 again as console, the following steps must be followed:
Start the system and stop the autoboot process by entering the appropriate key sequence (see III.3.1).
Then undefine the silent variable:

```
setenv silent
```

and set *bootcmd* and *bootargs* variables for NFS boot (see III.2.5). Save the environment and restart:
saveenv
reset

Once NFS boot is finished, login as root, mount the flash file system:

```
mount -t jffs2 /dev/mtdblock3 /mnt/local
```

Then edit /mnt/local/etc/inittab file to remove # at the beginning of "#S0:2345:respawn:/sbin/getty ttyS0 9600" line.

Restart the system and stop the autoboot to restore boot from flash (see III.2.6).

III.5 Software upgrades

III.5.1 U-boot upgrade

To install a new u-boot version, login as root and enter:

```
root@mbs270:~# dd if=my_new_u-boot_file of=/dev/mtdblock0 bs=256k
```

In standalone operation, you may transfer my_new_u-boot_file via TFTP.
With NFS, my_new_u-boot_file is stored somewhere in the file system like /data.
Note: be careful to respect the flash partition number! Or your MBS270 may become unusable. As long as u-boot partition (/dev/mtdblock0) is correct, it is possible to repair Linux and file system partitions via NFS boot (see III.2).

III.5.2 Linux kernel upgrade

To install a new Linux kernel, login as root and enter:

```
root@mbs270:~# dd if=my_new_Linux_kernel of=/dev/mtdblock2 bs=256k
```

In standalone operation, you may transfer my_new_Linux_kernel via TFTP.
With NFS, my_new_Linux_kernel is stored somewhere in the file system like /home/root.
Note: be careful to respect the flash partition number! Or your MBS270 may become unusable. As long as u-boot partition (/dev/mtdblock0) is correct, it is possible to repair Linux and file system partitions via NFS boot (see III.2).

III.5.3 File system upgrade

To install a new file system, boot MBS270 with NFS, login as root and enter:

```
root@mbs270:~# dd if=/data/new_file_system.jffs2 of=/dev/mtdblock3 bs=256k
```

assuming the new file system is stored in /data directory of the NFS file system.
This operation takes a few minutes because 28Mbytes are written in flash memory. It ends with the following messages:

```
110+0 records in
110+0 records out
```

that indicates 28Mbytes = 110 x 256kbytes have been written.
Note: be careful to respect the flash partition number! Or your MBS270 may become unusable. As long as u-boot partition (/dev/mtdblock0) is correct, it is possible to repair Linux and file system partitions via NFS boot (see III.2).

III.6 Kernel modules

Starting with August 2009 BSP, modules are installed the traditional way that is:
`/lib/modules/version/kernel/…`

For example, video acquisition related modules are in
`/lib/modules/version/kernel/drivers/media/video/`

Modules you wish to load at the end of the boot sequence shall be declared in /etc/modules. For example, the default content of this file for an MBS270 with a MBS7720 camera module is:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

# Modules for video acquisition with PXA270
pxa_camera


# Modules for MBS270 compatible cameras
# Uncomment only necessary modules

# MBS7720
soc_ov7720

# MBS032M
# pca953x
# mt9v022 sensor_type=monochrome

# MBS032C
# pca953x
# mt9v022 sensor_type=colour

# END Modules for MBS270 compatible cameras
```

III.7 Software packages, Angstrom distribution, ipkg

Many software packages can be downloaded at http://www.angstrom-distribution.org
Packages can be found in:

- http://www.angstrom-distribution.org/unstable/feed/all/
- http://www.angstrom-distribution.org/unstable/feed/armv5te and its subdirectories
- or browse by category: http://www.angstrom-distribution.org/repo

To install a package:
```
root@mbs270:~# ipkg install my_package.ipk
```

To remove a package:
```
root@mbs270:~# ipkg remove my_package
```

Enter `ipkg` command alone to see its options.

III.8 SD card operation

MBS270 V2 is equipped with a microSD card slot. User is encouraged to use this storage mean for intensive data write instead of onboard flash because it is simple and low cost to replace when flash memory chip runs out.
The MMC driver has been tested with SD type cards, not with SDHC ones.
Adding a line in /etc/fstab allows automatic mounting of the media when it is inserted or at boot time:

```
/dev/mmcblkp1     /media/card       auto  defaults,sync,noauto    0     0
```

Once operations on the card are finished, unmount it before ejecting it otherwise I/O errors may happen:

```
umount /media/card
```

Note: after pushing the card, remove it quickly to avoid spurious card detection.

III.8.1 I/O speed, synchronous vs asynchronous I/O

It is recommended to mount the media with asynchronous I/O (`async` option in /etc/fstab) to boost write speed as shown in the table below. These are average values obtained with `tests/test_sd` program and a 1GB Sandisk card. Different values may be obtained with alternate cards and buffer size.

	Synchronous mode	Asynchronous mode
Write speed	0.5MB/s	2.5MB/s
Read speed	40MB/s	12MB/s

III.9 RTC operation

MBS270 is equipped with an RTC chip that can save the current time when the board is powered down if the Lithium battery at G1 is connected.
The following paragraphs explain how to use this RTC by software.

III.9.1 Kernel configuration

To enable the RTC support in the kernel, several options must be set (with make menuconfig for example):

- CONFIG_RTC_CLASS
- CONFIG_RTC_LIB
- CONFIG_RTC_INTF_SYSFS for the sysfs interface
- CONFIG_RTC_INTF_PROC for the /proc interface
- CONFIG_RTC_INTF_DEV for the /dev interface
- CONFIG_RTC_DRV_DS1307 to enable the RTC chip driver

Notes:

- CONFIG_RTC_DRV_SA1100 shall not be set. This option enables the driver of the PXA270 internal RTC. It is possible to run with both RTCs but extra software should be added to avoid conflicts.
- CONFIG_RTC_HCTOSYS shall not be set. System clock is updated from hardware clock at boot time with init scripts (see below).

III.9.2 Hardware and system clocks updating

III.9.2.1 Basic commands

`date`	Shows the current system time.
`date –s MMDDhhmmYYYY.ss`	Changes the current system time.
`hwclock`	Shows the current time of the hardware clock pointed to by /dev/rtc.
`hwclock --hctosys`	Sets the current system time with the hardware clock.
`hwclock --systohc`	Sets the hardware clock with the current system time.

Be aware of the date in `/etc/timestamp` when changing times. This file must be deleted when setting the hardware clock with a time preceding the one indicated in `/etc/timestamp`.

III.9.2.2 Init scripts

The system time is set to the hardware clock time at boot up with a call to `/etc/init.d/hwclock.sh` in `/etc/init.d/bootmisc.h`.
The hardware clock is set by the system time at shutdown and reboot with a call to `/etc/init.d/save-rtc.sh`.

III.10 Utility programs

Several utility programs are provided for convenience.

III.10.1 mbs-gpio

`/usr/bin/mbs-gpio` is utility accessing the GPIO driver to provide GPIO access from command line. It can be used in standalone operation to:

- read or set the status of any available GPIO
- turn on or off LEDs D4 to D7 to indicate a status.
- turn on or off the 5V power supply
- read the status of S1 switches.

Aliases are defined in `/etc/profile` to simplify LEDs and switches access.

III.10.2 init_m41t0

`/usr/bin/init_m41t0` is a utility to initialize M41T0 real time clock (RTC) with the current system clock.

IV. IMAGE ACQUISITION

PXA270 is the first processor with XScale architecture including a hardware interface to directly connect a vision sensor. Called “Quick Capture Interface”, it supports DMA (Direct Memory Access) for CPU free image acquisition. Also, PXA270 includes Intel Wireless MMX technology; this enables high-performance multimedia acceleration with an industry proven instruction set.
Mobisense Systems has modified the driver developed by Guennadi Liakhovetski to capture images with the minimum acquisition delay and offer the best real time performance. This driver uses DMA and memory mapping for easy access to image buffer. The driver is V4L2 (Video For Linux 2) compatible.
V4L2 specifications make video capture under Linux simpler. For details about V4L2, please consult http://v4l2spec.bytesex.org.

IV.1 Image acquisition principle

The image sensor is connected to the Quick Capture Interface with a parallel connection. This makes possible uncompressed image acquisition at high data rate (up to 26Mbytes/s with MBS270 design).
When the user asks for an image, the request is stored by the driver and as soon as a frame arrives, it is transferred to system memory via DMA. The processor is free to do other tasks during the transfer such as processing a previously acquired image enabling real-time image processing. For example, it is possible to develop a multithreaded application with a vision thread and other threads. When the vision thread is running, it requests a frame, process the previous one and suspends until a new frame is available. In the mean time, others threads can execute.
The Quick Capture Interface is a video device; it is located at /dev/video0.
Access to image buffers is very easy with memory mapping: it is the same memory area where the driver stores a frame that is accessible by the user application for processing: no lengthy image copy to user space is required.
It is possible to acquire multiple frames successively. The maximum number of frame buffers depends of image size, pixel depth and available memory. Because MBS270 has 64MB of RAM, double buffer acquisition is possible with all kinds of sensors including multi megapixels.
Most image sensors can be configured via a serial interface, mostly I2C. The image acquisition driver performs I2C communication with the sensor to set up various acquisition parameters. By default, these parameters are set for optimal operation in common situation. It is possible to tune these parameters with standardized V4L2 user controls (V4L2_CID_xxx ioctl calls); this applies to image brightness, contrast, etc. It is also possible for advanced users to access sensor registers with V4L2 VIDIOC_DBG_G_REGISTER and VIDIOC_DBG_S_REGISTER ioctl calls.

IV.2 Video driver loading/unloading

The video driver is build as modules. The reason is once it is loaded, the Master Clock is turned on for the sensors. When unloaded, the Master Clock is off saving power.
Modules are detailed in section III.6.

IV.3 Image acquisition programming

The driver is fully V4L2 compatible so any compatible software can be used for image acquisition such as “Video Capture Example” in Appendix B of V4L2 specifications (http://v4l2spec.bytesex.org/spec/a16706.htm).
The following code can be found in /media/cdrom/BSP_*yymm*/app/tests/v4l2/test_grab.c example.
Any V4L2 image acquisition program must include `videodev2.h` file:

```
#include <linux/videodev2.h>
```

IV.3.1 Initialization (grab_init)

First open the device driver:

```
#define VIDEO_DEV_NAME "/dev/video0"
…

gctx->vfd = open(VIDEO_DEV_NAME, O_RDWR);
if (gctx->vfd < 0) {
  printf("%s ", VIDEO_DEV_NAME);
  perror("open failed");
  return -1;
}

printf("%s open succeeded.\n", VIDEO_DEV_NAME);
```

Then select the video input, only one so far:

```
input = 0;

ret = ioctl(gctx->vfd, VIDIOC_S_INPUT, &input);
if (ret != 0) {
  perror("ioctl VIDIOC_S_INPUT");
  return ret;
}
```

Then configure the acquisition format:

```
format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
format.fmt.pix.width = gctx->width;
format.fmt.pix.height = gctx->height;
format.fmt.pix.pixelformat = gctx->pixelformat;
format.fmt.pix.field = V4L2_FIELD_NONE;

ret = ioctl(gctx->vfd, VIDIOC_S_FMT, &format);
if (ret != 0) {
  perror("ioctl VIDIOC_S_FMT");
  return ret;
}
```

Then request buffers to store grabbed images:

```
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = gctx->nr_buffers;

ret = ioctl(gctx->vfd, VIDIOC_REQBUFS, &reqbuf);
if (ret < 0) {
  if (errno == EINVAL)
    printf ("Video capturing or mmap-streaming is not supported\n");
  else
    perror ("ioctl VIDIOC_REQBUFS");
  return ret;
}

if (reqbuf.count < gctx->nr_buffers) {
  printf("   !!! Only %d buffers available\n", reqbuf.count);
  gctx->nr_buffers = reqbuf.count;
}
```

Then query buffers to allocate memory and perform memory mapping, save mmap info for unmapping at end of program:

```
gctx->buffers = calloc(gctx->nr_buffers, sizeof (*gctx->buffers));
if (gctx->buffers == NULL) {
  perror("calloc gctx->buffers");
  return -1;
}

for (ii=0; ii<gctx->nr_buffers; ii++) {
  /* query buffer to allocate and mmap it */
  memset(&buffer, 0, sizeof (buffer));
  buffer.type = reqbuf.type;
  buffer.memory = V4L2_MEMORY_MMAP;
  buffer.index = ii;

  ret = ioctl(gctx->vfd, VIDIOC_QUERYBUF, &buffer);
  if (ret < 0) {
    perror ("ioctl VIDIOC_QUERYBUF");
    exit (EXIT_FAILURE);
  }

  printf("buffer %u: bytesused = %u\n",
         buffer.index, buffer.bytesused);

  gctx->buffers[ii].length = buffer.length; /* remember for munmap() */
  gctx->buffers[ii].start = mmap (NULL, buffer.length,
                           PROT_READ | PROT_WRITE,
                           MAP_SHARED,
                           gctx->vfd, buffer.m.offset);
  if (MAP_FAILED == gctx->buffers[ii].start) {
    perror ("mmap");
    return -1;
  }
}
```

Then image acquisition can start.

IV.3.2 Start image acquisition (grab_start)

Before acquiring images, all buffers must be queued and VIDIOC_STREAMON ioctl must be called:

```
memset(&buffer, 0, sizeof (buffer));
buffer.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buffer.memory = V4L2_MEMORY_MMAP;

for (ii=0; ii<gctx->nr_buffers; ii++) {
  buffer.index = ii;
  ret = ioctl(gctx->vfd, VIDIOC_QBUF, &buffer);
  if (ret < 0) {
    perror ("ioctl VIDIOC_QBUF");
    return ret;
  }

  printf("buffer %u: bytesused = %u\n", buffer.index, buffer.bytesused);
}

ret = ioctl(gctx->vfd, VIDIOC_STREAMON, &buffer.type);
if (ret < 0) {
  perror ("ioctl VIDIOC_STREAMON");
  return ret;
}

printf("Wait for sensor auto adjust...");
fflush(stdout);
sleep(1);
printf(" Done.\n");
```

When called, VIDIOC_STREAMON ioctl wakes up the sensor so it is better to wait a few frames for sensor auto adjust before starting image processing.

Dequeue the first images:

```
for (ii=0; ii<gctx->nr_buffers; ii++) {
  /* retrieve image when sensor turned on */
  buffer.index = ii;
  ret = ioctl(gctx->vfd, VIDIOC_DQBUF, &buffer);
  if (ret < 0) {
    if (errno == EAGAIN)
      printf("Image not grabbed yet.\n");
    else
      perror("ioctl VIDIOC_DQBUF");
    return ret;
  }
}
```

IV.3.3 Acquisition (grab_image)

Images are acquired with VIDIOC_QBUF and VIDIOC_DQBUF ioctl calls:

```
for (ii=0; ii<gctx->nr_buffers; ii++) {
  buffer.index = ii;
  ret = ioctl(gctx->vfd, VIDIOC_QBUF, &buffer);
  if (ret < 0) {
    perror ("ioctl VIDIOC_QBUF");
    goto stream_off;
  }
  printf("buffer %u: bytesused = %u\n", buffer.index, buffer.bytesused);
}

/* dequeue buffers */
for (ii=0; ii<gctx->nr_buffers; ii++) {
  buffer.index = ii;
  ret = ioctl(gctx->vfd, VIDIOC_DQBUF, &buffer);
  if (ret < 0) {
    if (errno == EAGAIN)
      printf("Image not grabbed yet.\n");
    else
      perror("ioctl VIDIOC_DQBUF");
    goto stream_off;
  }
  printf("%ld.%06ld DQBUF %02d end, frame %u.\n",
         buffer.timestamp.tv_sec, buffer.timestamp.tv_usec,
         buffer.index, buffer.sequence);
}
```

IV.3.4 Stop image acquisition (grab_stop)

Image acquisition is stopped with VIDIOC_STREAMOFF ioctl:

```
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

ret = ioctl(gctx->vfd, VIDIOC_STREAMOFF, &type);
if (ret < 0) {
  perror ("ioctl VIDIOC_STREAMOFF");
  return ret;
}
```

When called, VIDIOC_STREAMOFF ioctl turns the sensor into sleep mode.

IV.3.5 End image acquisition (grab_end)

At end of image acquisition, memory mapped buffers must be unmapped and video device closed:

```
int ii;
if (gctx->buffers != NULL) {
  for (ii=0; ii<gctx->nr_buffers; ii++) {
    if ((gctx->buffers[ii].start != MAP_FAILED) &&
        (gctx->buffers[ii].length > 0))
      munmap (gctx->buffers[ii].start, gctx->buffers[ii].length);
  }

  free(gctx->buffers);
}

if (gctx->vfd != -1) close(gctx->vfd);
```

IV.4 The mbs270_demo program

This program demonstrates MBS270 operating as a video server. The client is a Windows based application that sends requests to the MBS270 and displays images received in answers. This program shows the various image formats MBS270 can acquire with a camera sensor.

IV.4.1 Installing and starting

The server is preinstalled on MBS270 in the /home/root/demos directory. After login as root, launch the server with the following commands:

```
$ /usr/local/video start my_camera_module
$ ./demos/mbs270_demo_srv
```

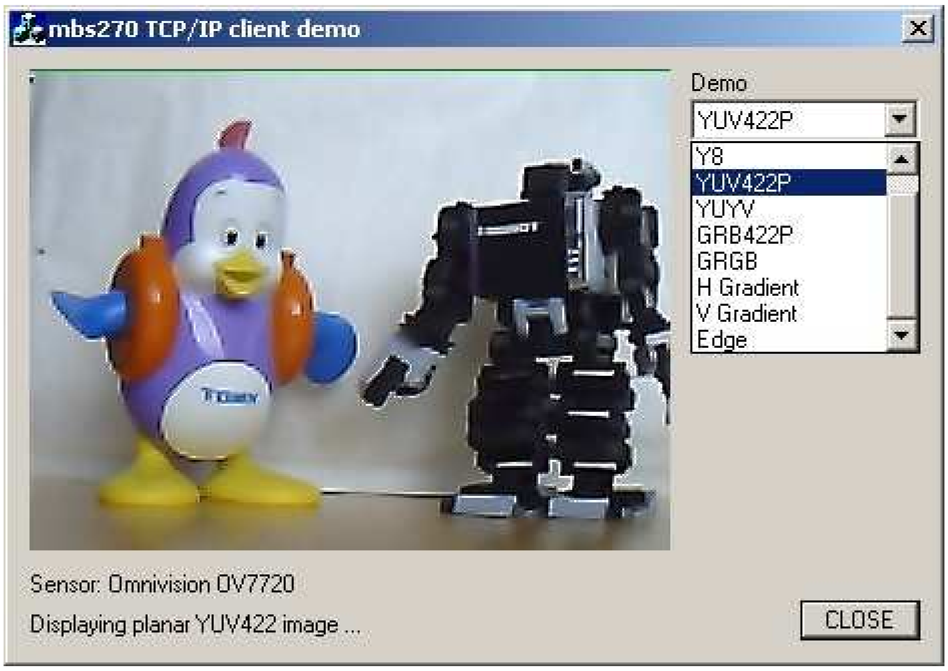
The client shall be installed on a Windows based computer by copying the entire app/demos/bin directory. An Ethernet connection must be established between the Windows PC and MBS270. The client can be launched in a command window with the following command :

```
C:\...>Client_demo2.exe mbs270_IP_address
```

or a shortcut can be created with this command.

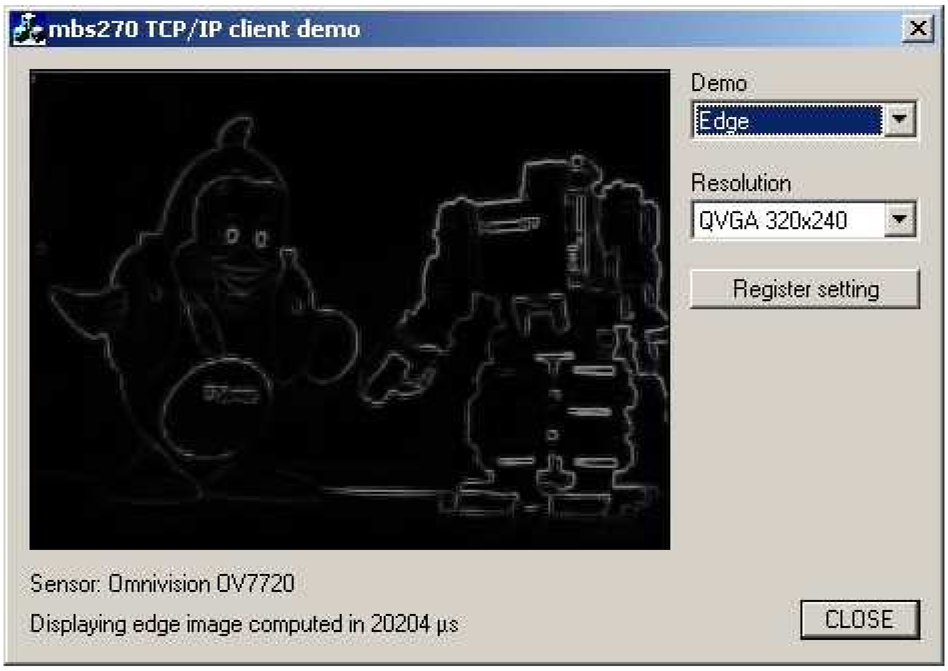
IV.4.2 Demo content

The client displays images acquired and/or computed by MBS270 in various formats. Available formats depend of the camera module connected to MBS270 so the “Demo” drop down list is dynamically filled according to the module type. One can not expect colored images with a monochrome sensor or gradient images with a Bayer only sensor. Similarly, resolution is sensor dependent and the “Resolution” drop down list is dynamically filled.
The following screen shots were obtained with a MBS7720 camera module. The OV7720 sensor it contains can provide both color and monochrome images so many formats are available as shown in the “Demo” list. Also the sensor type is indicated under the image as well as the current status.



Sensor: Omnivision OV7720
Displaying planar YUV422 image ...

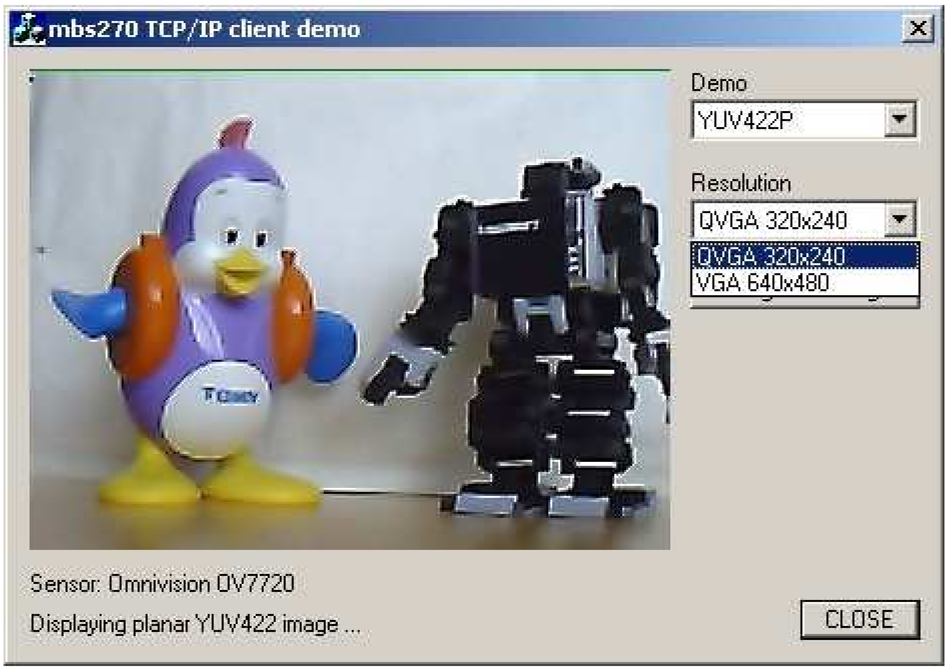
Select an alternate image format from the drop down list in two steps: first select Standby to stop current acquisition then select new format to enable the server to reconfigure image acquisition.



Demo: Edge
Resolution: QVGA 320x240

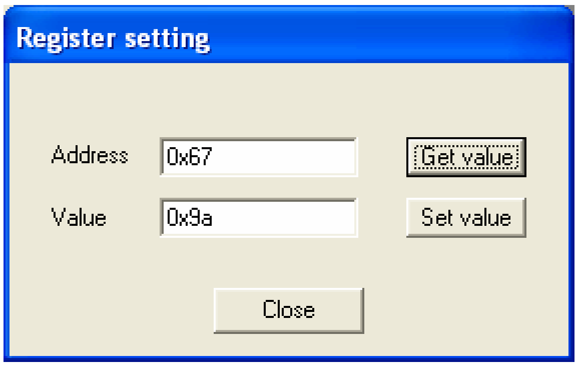
Sensor: Omnivision OV7720
Displaying edge image computed in 20204 µs

The resolution can be changed with the Resolution list. Note that image update is slower with VGA or higher resolution due to network bandwidth.



Sensor: Omnivision OV7720
Displaying planar YUV422 image ...

Press the *Register setting* button to access to the register dialog. It is possible to read current sensor register values and to modify them. **Use this function only if you have the sensor registers definition and are aware of the effects of the modifications you do!**



Address: 0x67
Value: 0x9a

V. ROBOTICS DEDICATED FEATURES

V.1 Preamble

Linux installed in MBS270 contains specific drivers to offer interfaces convenient for robotics applications: GPIO, PWM, Radio command servo, timer and ADC.
Software interface for these drivers is described in the following sections. Most of the interface relies on ioctl calls, not read or write calls. This is a design choice motivated by the fact that such devices are not frequently used individually. Ioctl calls allow programming of a set of GPIO or PWM simultaneously.

V.2 GPIO control

The GPIO driver allows a user application to set/unset GPIO outputs and read state of GPIO inputs. This can be done with a generic open of /dev/mem and mmap call but using a driver makes things more robust.
Internally, the GPIO driver manages GPIO use between applications and other drivers: PWM, R/C servo, timer and others. It is not possible for an application to take control over a GPIO used by another application. It is neither possible for an application to use a GPIO as an RC servo output for example if it is already used for another purpose: GPIO, PWM or internally for example.
/usr/bin/mbs-gpio is a utility program accessing this driver (see III.10.1).

Important note:
The Linux kernel 2.6.26 includes a generic GPIO access library to avoid concurrent use of GPIOs. MBS270 GPIO driver has been adapted to run with this library. It is no more PXA specific so its device name has been changed from /dev/pxa-gpio to **/dev/gpio**.
The major number is 10 (miscellaneous character device) and the minor number is set dynamically.

Note: this driver should not be used to generate periodic outputs at high frequency. Use PWM or timer outputs described further instead.

V.2.1 GPIO programming tutorial

The following code comes from /media/cdrom/BSP_*yymm*/app/tests/gpio/test_gpio.c test program.

Two include files shall be included to access the GPIO driver:

```
#include <linux/ioctl.h>
#include <linux/gpiodev.h>          // formerly <linux/pxa_gpio.h>
```

Open the GPIO driver as follow:

```
  gpio_dev = open(GPIO_DEV_NAME, O_RDWR);
  if (gpio_dev < 0) {
    printf("%s ", GPIO_DEV_NAME);
    perror("open");
    exit(1);
  }
```

with GPIO_DEV_NAME defined as:

```
#define GPIO_DEV_NAME "/dev/gpio"  // formerly /dev/pxa-gpio
```

Register a GPIO output with IOCGPIODEVADD ioctl call:

```
  gpio_add.gpio = (unsigned int) atoi(argv[1]);
  gpio_add.input = 0;
  gpio_add.init_state = 1;
  gpio_add.permanent = 0;
  if (ioctl(gpio_dev, IOCGPIODEVADD, (void*) &gpio_add)) {
    perror("ioctl IOCGPIODEVADD: ");
  }
```

This call requires a pointer to a struct gpiodev_add variable that defines the GPIO number (gpio field), its direction (input field, 0 means output, non null value means input), its initial output state (init_state field) and the permanent status. Permanent status is for outputs only. By default, all requested GPIOs are set as inputs when closing the driver. However, it is possible to force a GPIO to remain as output by setting the permanent flag.

Registering a GPIO input is similar except that the init_state field is initialized by the driver during the call.

A GPIO output state is set with IOCGPIODEVSET ioctl call:

```
if (ioctl(gpio_dev, IOCGPIODEVSET, (void*) &gpio_add.gpio))
  perror("ioctl IOCGPIODEVSET: ");
```

A GPIO output state is cleared with IOCGPIODEVCLR ioctl call:

```
if (ioctl(gpio_dev, IOCGPIODEVCLR, (void*) &gpio_add.gpio))
  perror("ioctl IOCGPIODEVCLR: ");
```

A GPIO input state is retrieved with IOCGPIODEVGET ioctl call:

```
if (ioctl(gpio_dev, IOCGPIODEVGET, (void*) &gpio_get))
  perror("ioctl IOCGPIODEVGET: ");
```

This call requires a pointer to a struct gpiodev_get variable that defines the GPIO number (gpio field) and the state (state field, value = 0 or 1).

When a GPIO is not useful anymore, it is released with IOCGPIODEVREM ioctl call:

```
if (ioctl(gpio_dev, IOCGPIODEVREM, (void*) &gpio_add.gpio)) {
  perror("ioctl IOCGPIODEVREM: ");
  goto close_gpio_device;
}
```

V.2.2 GPIO programming reference

Ioctl call	Effect	Parameter	Return value
IOCGPIODEVADD	Reserve a GPIO for an application as input or output.	struct gpiodev_add * gpio = GPIO number as defined in MBS270 hardware manual, input = 0 for output, not 0 for input, init_state = initial state set by the application for outputs, by the driver for inputs. Permanent = 0 GPIO is input on close, not 0 output can be maintained.	0 = successful -EINVAL = invalid GPIO number -EBUSY = GPIO in use by system or application
IOCGPIODEVSET	Set a GPIO output.	Int * Points to GPIO number previously requested with IOCGPIODEVADD ioctl call.	0 = successful -EINVAL = invalid GPIO number or not GPIO owner or GPIO is not an output.
IOCGPIODEVCLR	Clear a GPIO output.	Int * Points to GPIO number previously requested with IOCGPIODEVADD ioctl call.	0 = successful -EINVAL = invalid GPIO number or not GPIO owner or GPIO is not an output.
IOCGPIODEVGET	Get a GPIO input state.	struct gpiodev_get * gpio = GPIO number previously requested with IOCGPIODEVADD ioctl call, state = input state read by the driver.	0 = successful -EINVAL = invalid GPIO number or not GPIO owner or GPIO is not an input.
IOCGPIODEVREM	Release a GPIO for other use.	Int * Points to GPIO number previously requested with IOCGPIODEVADD ioctl call.	0 = successful -EINVAL = invalid GPIO number -EBUSY = GPIO in use by system or application

In case of error, check MBS270 hardware manual for GPIO number and availability.

V.3 PWM outputs

PWM outputs provide a basic digital to analog converter with an appropriate analog filter. They can also be used to drive motor speed controllers.
The PWM driver allows a user application to interact with PXA270 PWM controller. As for GPIOs, this can be done with a generic open of /dev/mem and mmap call but using a driver makes things more robust.
The PWM device is /dev/pxa-pwm. Its major number is 10 (miscellaneous character device) and its minor number is set dynamically.

V.3.1 PWM programming tutorial

The following code comes from /media/cdrom/BSP_*yymm*/app/tests/pwm/test_pwm.c test program.

Two include files shall be included to access the PWM driver:

```
#include <linux/ioctl.h>
#include <linux/pxa_pwm.h>
```

Open the PWM driver as follow:
```
  pwm_dev = open(PWM_DEV_NAME, O_RDWR);
  if (pwm_dev < 0) {
    perror("open");
    exit(1);
  }
```

with PWM_DEV_NAME defined as:
```
#define PWM_DEV_NAME "/dev/pxa-pwm"
```

Register a PWM output with PXAIOCPWMADD ioctl call:
```
  pwm_add.nr = (unsigned int) atoi(argv[1]);
  pwm_add.f_hz = (unsigned int) atoi(argv[2]);
  pwm_add.duty = atoi(argv[3]);
  if (ioctl(pwm_dev, PXAIOCPWMADD, (void*) &pwm_add)) {
    perror("ioctl PXAIOCPWMADD: ");
  }
```

This call requires a pointer to a struct pxa_pwm_add variable that defines the PWM number between 0 and 3 (nr field), the signal frequency in Hertz (f_hz field) and its initial duty cycle between 0 and 1023 (duty field).

Duty cycle of PWM outputs is modified with PXAIOCPWMDUTY ioctl call:
```
  if (ioctl(pwm_dev, PXAIOCPWMDUTY, (void*) &pwm_chg)) {
    perror("ioctl PXAIOCPWMDUTY: ");
  }
```

This call requires a pointer to a struct pxa_pwm_chg variable that defines for each concerned PWM output its number between 0 and 3 (nr field) and its duty cycle between 0 and 1023 (duty field).

When a PWM is not useful anymore, it is released with PXAIOCPWMREM ioctl call:
```
  if (ioctl(pwm_dev, PXAIOCPWMREM, (void*) &pwm_add.nr)) {
    perror("ioctl PXAIOCPWMREM: ");
  }
```

V.3.2 PWM programming reference

Ioctl call	Effect	Parameter	Return value

PXAIOCPWMADD	Register a PWM output.	struct pxa_pwm_add *
		nr = PWM number between 0 and 3,
		f_hz = PWM frequency in Hertz,
		duty = initial PWM duty cycle: 0 = 0%, 1023 = 100%.
			0 = successful
			-EINVAL = invalid PWM number
			-EBUSY = PWM in use by system or application

PXAIOCPWMDUTY	Modify PWM duty cycle.	struct pxa_pwm_chg * nr_chg = number of duty cycles to change For each change: nr = PWM number between 0 and 3, duty = initial PWM duty cycle: 0 = 0%, 1023 = 100%.	0 = successful -EINVAL = invalid PWM number.
PXAIOCPWMREM	Release a PWM output.	Int * Points to PWM number previously registered with PXAIOCPWMADD ioctl call.	0 = successful -EINVAL = invalid PWM number

V.3.2.1 PWM Frequency range

PWM frequency is derived from a 13MHz internal clock. This frequency can be divided up to 64 times. For 10 bits resolution, the PWM minimum frequency is 13MHz/(64*1024) = 198,4Hz and the maximum frequency is 13MHz/1024 = 12,7 kHz.
Frequency below 198,4Hz is not possible.
If higher frequency is required, only lower resolution can be achieved as shown in the table below:

PWM resolution (bits)	Frequency (kHz)
10	0.2 to 12.7
9	25.4
8	50.8
7	101.6
6	203.1
5	406.2
4	812.5
3	1625.0
2	3250.0
1	6500.0

V.4 R/C servo outputs

Radio command servo motors are electromechanical systems developed for R/C vehicles and commonly used in robotics systems. Servos are a simple and efficient solution to control rotation about one axis: they can be found in walking robots or any articulated system.
A servo is made of a DC motor, a gear box, a potentiometer mounted on the output shaft that provides position feedback and a controller. A servo is electrically connected with three wires: power, ground and signal.
Signal is an input that receives periodical pulses which duration sets the output shaft position. Typical values are 1ms duration for -90°, 1.5ms for neutral position 0° and 2ms for +90°, but this may vary from one manufacturer to another (please consult websites of manufacturers like Futaba, Hitec or Graupner for more details). For most servos, pulse duration accuracy must be better than 10 microseconds to guarantee 1° accuracy of output shaft position.
Pulse must be sent to Signal input at least every 20ms or the controller goes idle to save power.
MBS270 servo driver reproduces this timing for up to 32 GPIOs with less than 1 microsecond accuracy. Pulse duration can be defined between 5 and 3000 microseconds (see pxa_servo.h) to allow control of almost any R/C servo. The driver uses Fast Interrupt Request (FIQ) mechanism of PXA270 processor.
The R/C servo device is /dev/pxa-servo. Its major number is 10 (miscellaneous character device) and its minor number is set dynamically.

Note: unlike PWM or timer outputs, servo signals are generated by software and require a small amount of CPU time. This driver is convenient to generate time accurate outputs for many servos. In an application with few servos, it is better to use PWM outputs if they are available and accurate enough.

V.4.1 Servo programming tutorial

The following code comes from /media/cdrom/BSP_*yymm*/app/tests/servo/test_servo.c test program.

Three include files shall be included to access the R/C servo driver:
```
#include <linux/ioctl.h>
#include <linux/pxa_servo.h>
#include <linux/miscdevice.h>
```

Open the R/C servo driver as follow:
```
  servo_dev = open(SERVO_DEV_NAME, O_RDWR);
  if (servo_dev < 0) {
    printf("%s ", SERVO_DEV_NAME);
    perror("open failed");
    exit(1);
  }
```

with SERVO_DEV_NAME defined as:
```
#define SERVO_DEV_NAME "/dev/pxa-servo "
```

Register a servo output with PXAIOCSRVADD ioctl call:
```
  if (ioctl(servo_dev, PXAIOCSRVADD, (void*) &srv_add)) {
    perror("ioctl PXAIOCSRVADD: ");
  }
```

This call requires a pointer to a struct pxa_servo_add variable that defines the GPIO number used to generate the pulse (gpio field), the pulse duration in microseconds (delay field) and the servo output identifier used in further ioctl calls (id field). The first two fields are set by the application before call; the identifier is set by the driver during call.

Pulse duration of servo outputs is modified with PXAIOCSRVCHG ioctl call:
```
  if (ioctl(servo_dev, PXAIOCSRVCHG, (void*) &srv_chg))
    perror("ioctl PXAIOCSRVCHG: ");
```

This call requires a pointer to a struct pxa_servo_chg variable that defines the number of outputs to change (nb field) and a list of changes, each defined by the identifier (id field) and the pulse duration in microseconds (delay field).

Unregister a servo output with PXAIOCSRVREM ioctl call:
```
  if (ioctl(servo_dev, PXAIOCSRVREM, (void*) &srv_rem))
    perror("ioctl PXAIOCSRVREM: ");
```

This call requires a pointer to a struct pxa_servo_rem variable that defines the servo output identifier (id field).

V.4.2 Servo programming reference

Ioctl call	Effect	Parameter	Return value
PXAIOCSRVADD	Register a servo output.	struct pxa_srv_add * gpio = GPIO number used to generate the pulse, delay = pulse duration in microseconds, id = servo output identifier.	0 = successful -EINVAL = invalid GPIO or delay -EBUSY = GPIO in use or no more servo available
PXAIOCSRVCHG	Modify pulse duration of servo outputs.	struct pxa_srv_chg * nb = number of servos to change For each change: id = servo output identifier, delay = pulse duration in microseconds.	0 = successful -EINVAL = servo identifier or pulse duration.
PXAIOCSRVREM	Release a servo output.	struct pxa_srv_rem * id = servo output identifier.	0 = successful -EINVAL = invalid servo identifier.

V.5 Timer outputs

Timer outputs provide a way to generate periodic outputs derived from the Operating System Timers block of the PXA270 processor. The highest frequency a timer output can reach is 250 kHz. The lowest frequency is about 2.33*10^{-7} Hz. So these channels can be used for fast or very slow clocks. Timer outputs have a fixed duty cycle equal to 0.5.
The timer device is /dev/pxa-timer. Its major number is 10 (miscellaneous character device) and its minor number is set dynamically.
As for GPIOs and PWMs, the timer block registers can be accessed with a generic open of /dev/mem and mmap call but using a driver makes things more robust.

V.5.1 Timer programming tutorial

The following code comes from /media/cdrom/BSP_*yymm*/app/tests/timer/test_timer.c test program.

Two include files shall be included to access the R/C servo driver:
```
#include <linux/ioctl.h>
#include <linux/pxa_timer.h>
```

Open the timer driver as follow:
```
  timer_dev = open(TIMER_DEV_NAME, O_RDWR);
  if (timer_dev < 0) {
    perror("open");
    exit(1);
  }
```

with TIMER_DEV_NAME defined as:
```
#define TIMER_DEV_NAME "/dev/pxa-timer "
```

Register a timer channel output with PXAIOCTMR_ADDCO ioctl call:
```
  if (ioctl(timer_dev, PXAIOCTMR_ADDCO, (void*) &timer_add)) {
    perror("ioctl PXAIOCTMR_ADDCO: ");
  }
```

This call requires a pointer to a struct pxa_timer_add_chout variable that defines the timer channel (0 or 1, nr field), the frequency in Hertz (f_hz field) and a divider for sub Hertz frequencies (ratio field). The output frequency is f_hz / ratio.

Frequency can be modified with the PXAIOCTMR_FREQCO ioctl call:

```
  if (ioctl(timer_dev, PXAIOCTMR_FREQCO, (void*) &freq_chout)) {
    perror("ioctl PXAIOCTMR_FREQCO: ");
    bcontinue = 0;
    break;
  }
```

This call requires a pointer to a struct pxa_timer_freq_chout variable that defines the timer channel (0 or 1, nr field), the frequency in Hertz (f_hz field) and a divider for sub Hertz frequencies (ratio field). The output frequency is f_hz / ratio.

Unregister a timer channel output with PXAIOCTMR_REMCO ioctl call:

```
  if (ioctl(timer_dev, PXAIOCTMR_REMCO, (void*) &timer_add.nr)) {
    perror("ioctl PXAIOCTMR_REMCO: ");
  }
```

V.5.2 Timer programming reference

Ioctl call	Effect	Parameter	Return value

PXAIOCTMR_ADDCO	Register a timer channel output.	struct pxa_timer_add_chout *
		nr = timer channel: 0 or 1,
		f_hz = frequency in Hertz,
		ratio = frequency divider for sub Hertz output.
			0 = successful
			-EINVAL = invalid parameter
			-EBUSY = GPIO or timer channel in use

PXAIOCTMR_FREQCO	Modify frequency of timer channel output .	struct pxa_timer_freq_chout *
		nr = timer channel: 0 or 1,
		f_hz = frequency in Hertz,
		ratio = frequency divider for sub Hertz output.
			0 = successful
			-EINVAL = invalid parameter.

PXAIOCTMR_REMCO	Release a timer channel output.	int * pointer to timer channel.	0 = successful -EINVAL = invalid timer channel.

V.6 Analog to digital converter inputs

Four analog inputs are available on MBS270 thanks to UCB1400 companion chip on Colibri module (there is no analog input on the PXA270 processor itself). Data is transmitted from UCB1400 chip to PXA270 via an AC’97 link.
The analog to digital converter device is /dev/ucb1400-adc (formerly /dev/adconv/ucb1x00). Its major number is 10 (miscellaneous character device) and its minor number is set dynamically.

Note: for better comprehension and compliance with Linux standards, the device has been renamed /dev/ucb1400-adc and declarations have been moved from <linux/miscdevice.h> to a dedicated file <linux/ucb1400_adc.h>.

V.6.1 ADC programming tutorial

The following code comes from /media/cdrom/BSP_*yymm*/app/tests/adc/test_adc.c test program.

Two include files shall be included to access the ADC driver:

```
#include <linux/ioctl.h>
#include <linux/ucb1400_adc.h>     // formerly <linux/miscdevice.h>
```

Open the ADC driver as follow:

```
  adc_dev = open(ADC_DEV_NAME, O_RDWR);
  if (adc_dev < 0) {
    perror("open");
    exit(1);
  }
```

with TIMER_DEV_NAME defined as:

```
#define ADC_DEV_NAME "/dev/ucb1400-adc"  // formerly "/dev/adconv/ucb1x00"
```

ADC channels can be read with MISCIOCADC ioctl call. Up to four channels can be read simultaneously:

```
  struct misc_adc adc_data;
  adc_data.channel_count = 4;
  adc_data.channel[0].number = 0;
  adc_data.channel[1].number = 1;
  adc_data.channel[2].number = 2;
  adc_data.channel[3].number = 3;
  if (ioctl(adc_dev, MISCIOCADC, (void*) &adc_data) < 0) {
    perror("ioctl ADC");
  }
```

Conversion results are stored by the driver in respective adc_data.channel[xx].value fields.

There is no protection on these inputs: any application can access them because they do not have any specific configuration as shown in test_adc_mt.c multithreaded example. Of course, the driver manages sequential access to the ADC which is unique resource of the system.

V.6.2 ADC programming reference

Ioctl call	Effect	Parameter	Return value
MISCIOCADC	Read Analog to digital converter inputs	struct misc_data * count = number of channels to read. For each channel: number = ADC channel: 0 to 3, value = conversion result set by the driver.	0 = successful -EINVAL = invalid value for count or number

V.6.3 ADC speed performance

Although an ADC conversion takes about 10 microseconds on the UCB1400, it takes much longer to retrieve a value with software because of AC'97 link interface.
Conversion speed can be evaluated with the sample program:
/media/cdrom/BSP_*yymm*/app/tests/adc/test_adc_speed.c.

This program gives an estimate of the conversion time:

Number of channels	Call syntax	Execution time for 1000 conversions (ms)	Estimated conversion time (µs)
1	`test_adc_speed 1`	279	279
2	`test_adc_speed 2`	470	235
3	`test_adc_speed 3`	630	210
4	`test_adc_speed 4`	800	200

These results show that conversion time is lower for multiple channels. When multiple channels are acquired, it is recommended to do a single call with the list of all channels instead of one call per channel.