Jürgen Beisert

GeoTerm Quickstart

DRAFT – DRAFT – DRAFT – DRAFT

Rev : 809 $Date$: $2007-10-0612:38:54+0200(Sat, 06Oct2007)$

Contents

Part I

Quickstart for Igel GeoTerm

1 Some notes about GeoTerm

The compiler mentioned in chapter 2.3 is not part of the OSELAS-Toolchain package (but you will need this package anyway).
Instead you will find it in this project. So when running the `ptxdist select` command you should use the `i586-pmmx-linux-gnu_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig` file from this project instead.

1.1 About the Project

The goal of this project was to get a small powerful Linux based X terminal. It should be as silent as possible (that was no problem, because no fans!), but should also play music and desktop sounds.

As I'm using more than one of this small machines everyone in my family should use its own machine. But every local setup differs. Two setups using a SXGA TFT monitor, one a XGA TFT monitor, and another one an older SXGA like CRT monitor. And there are various mice and keyboards in use (PS/2 and USB based).

But I want run them all from one setup. They all booting their kernel via TFTP and using an NFS rootfilesystem and get their information what to boot, where to mount from a DHCP server. But these machines are terminals only. After the Xorg server is up and running they connecting itself via XDM to my main workstation and all the large KDE applications are running there. A 300MHz Geode processor can't run such large applications itself. Its to slow. But working as a simple terminal is the right job for it.

This board support package brings all the required parts to get this terminal to work.

1.2 About the GeodeGX1 Processor

The GeodeGX1 processor is a Pentium MMX derivate. Its an old device and you cannot buy it anymore. But it was very successfull, built into many applications like terminals and other embedded systems.

Its most known feature is the extensive use of the *System Management Mode* to emulate hardware that doesn't exist on this platform. A GeodeGX1 based system is like a standard PC, but not really. It support VGA graphics and sound. But this hardware is not compatible with other VGA and sound hardware.

For example the VGA hardware cannot handle text mode. It only supports graphic modes. Thanks to the SMM it emulates text mode transparent to the software. Also the sound hardware has its own implementation. To avoid adapting every software the SMM emulates a soundblaster device. So its possible to reuse the old drivers for the soundblaster device.

But this extensive use of the *System Management Mode* was also the reason why many people dammed it: Emulation is very slow. If this processor was used in a real time environment (Yes! Many people did so!) the SMM was always one reason to become desperate. The serial console does not work correctly and looses chars even on 9600Bd if the text screen scrolls one line! The SMM has the highest priority in this system, so every real time OS looses control.

But there are also advantages of this processor: Its low power consumption. Everytime the CPU runs into a HALT instruction it stops the clock. If the CPU is only frequently used the device becomes warm, not hot! This feature is very nice in combination with dynamic ticks, so the clock stops as long as ever possible.

1.3 About LinuxBios

To get rid of the awful SMM I replaced the standard BIOS with LinuxBIOS. LinuxBIOS is mostly a hardware configurator and initialiser. After this is done, it starts a so called payload. The payload does the rest to bring up the system as you like. There are different payloads available. It depends on the requirements what payload is the right one. I need support to boot from network or from local harddisk. So my payloads are Etherboot and FILO.

The disadvantage of LinuxBIOS and its lack of SMM support is, for each hardware feature you want to use you need a specific driver! You cannot use a Sound-Blaster driver, if there is no Sound-Blaster emulation present! But with a specific driver you will win performance. For example on a 233MHz Geode with Sound-Blaster emulation to play internet radio it consumes about 50% of the CPU. With a native sound driver to play an MPG3 audio it only consumes 5% of the CPU!

Also without the SMM you cannot use the standard X11 graphic driver. It expects the SMM and relies on it. I have no clue why this hardware specific driver uses the emulated registers (for example to save and restore the VGA registers). But with LinuxBIOS in the background this official driver fails.

1.4 About the Hardware

The basic hardware was a Linux based graphic terminal from IGEL type 316. It was shipped with some kind of BIOS replacement (I wasn't able to enter the BIOS in a way like a standard PC) and boots everything from the internal 16MiB flash disk. It takes about a minute to boot this device, I don't know why. It starts with a windows like desktop with only one application: A configurator. Within this application you where able to configure the device and the connection to other hosts (Windows- and Unix hosts). But I wasn't able to save the settings, so after each reboot, I had to reenter everything. When I started the automatic update process, it failed and the device ends up in a standard console. Very nice. That's why I know they are running a 2.4 kernel with a 3.3 Xfree server.

Now I'm running a 2.6 kernel and Xorg 7.2 and the device boots up in about 10 seconds into a remote xdm dialouge.

1.5 What else?

To replace the standard BIOS with LinuxBIOS you will need a flash device programmer. I built one by my own. A small and ugly one but it does its job.

1.6 LinuxBios

Building the LinuxBios for this platform is part of the BSP. After building the whole project we can find the ready-to-flash binary image in `images/linuxbios.rom`.

The ptxdist-1.0.0ensure building etherboot first to automatically include it into the LinuxBIOS.

Note: This binary image relies on the IGEL316 hardware. We cannot us it on any other GX1 based hardware. At least the interrupt routing differs so this binary image will fail at runtime!

Refer `patches/LinuxBIOSv2-r2671/generic` for the IGEL316 patch set to let LinuxBIOS know it.

1.7 EtherBoot

The required Etherboot payload needed by LinuxBIOS will also build within this project. No further user intervention is required.

This etherboot is configured for the IGEL316 hardware (mostly its ethernet device). Using this etherboot on different platform will fail at runtime.

Refer `patches/etherboot-5.4.2/generic` for the IGEL316 patch set to configure etherboot.

1.8 Linux Kernel

Building the kernel is part of this BSP. Currently a 2.6.22-8-cfs will be build.

All required patches to support this GX1 system are included in this BSP (refer `patches/linux-2.6.22/generic` for the patch set).

The kernel configuration can be found in the `igel316-kernelconfig.target` file. Do not modify things manually in this file, use always the *correct* way:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2$ ptxdist kernelconfig
```

After building the whole project we can find the plain kernel in `images/linuximage`. As we cannot load such a plain kernel with etherboot a second step appends this kernel with etherboot related information and store it into several `images/*_kernel` files.

The names for these kernels are predefined (for my environment). Defining the kernel command line for these kernels happens in the menu configuration and creating each of them happens in `rules/create_various_kernel.make`.

1.9 Hardware found in my IGEL 316

It seems this system bases on a BCOM WINNET100 system.

Some hardware description can be found at

http://www.linuxbios.org/index.php/BCOM_WINNET100_Build_Tutorial

Here is a list of devices:

GX1-300B-85-20 Cyrix/National/AMD Main CPU with 300MHz core clock

CS5530A-UCE part of the chipset, Cyrix/National/AMD companion device

PC97317 Super IO

RTL8139C Realtec network controller

39F020A 256kiB PLCC32 Flash memory to boot

LM4546 National, AC97 AD/DA

DOC200 16MiB DIL32 DiskOnChip

SDRAM 32MiB SDRAM with 133MHz/CL2 capability

The small memory makes it useless as a surf station. X uses local memory to cache pictures when you visit websites. Some websites forces the X server to use more than 50MiB of memory. When this happens the local system ooppses and sometimes kills the X server only, sometimes the whole system freezes. At least 64MiB seems a usefull memory size. I run one system with 32MiB, one with 64MiB and two systems with 128MiB memory.

2 Getting a working Environment

2.1 Download Parts

In order to follow this manual, some software archives are needed. There are several possibilities how to get these: either as part of an evaulation board package, or by download from a world wide web site.

The central place for OSELAS related documentation is http://www.oselas.com. This website provides all required packages and documentation (at least for software components which are available to the public).

To build OSELAS.BSP-JB-GeoTerm-2, the following archives should be available on the development host:

- ptxdist-1.0.0.tgz
- ptxdist-1.0.0-patches.tgz
- OSELAS.BSP-JB-GeoTerm-2.tgz
- OSELAS.Toolchain-1.1.0.tgz

2.2 PTXdist Installation

PTXdist is shipped divided into several archives. This chapter provides information about how to install and configure PTXdist on the development host to get a working environment to build root filesystems for target systems.

2.2.1 Building Blocks

The main tool of the OSELAS.BoardSupport() Package is PTXdist. So before starting any work we'll have to install PTXdist on the development host. PTXdist consists of the following parts:

The `ptxdist` Program: `ptxdist` is installed on the development host during the installation process. `ptxdist` is called to trigger any action, like building a software packet, cleaning up the tree etc. Usually the `ptxdist` program is used in a *workspace* directory, which contains all project relevant files.

A Configuration System: The config system is used to customize a *configuration*, which contains information about which packages have to be built and which options are selected.

Patches: Due to the fact that some upstream packages are not bug free – especially with regard to cross compilation – it is often necessary to patch the original software. PTXdist contains a mechanism to automatically apply patches to packages. The patches are bundled into a separate archive. Nevertheless, they are necessary to build a working system.

Package Descriptions: For each software component there is a "recipe" file, specifying which actions have to be done to prepare and compile the software. Additionally, packages contain their configuration sniplet for the config system.

Toolchains: PTXdist does not come with a pre-built binary toolchain. Nevertheless, PTXdist itself is able to build toolchains, which are provided by the OSELAS.Toolchain() project. More in-deep information about the OSELAS.Toolchain() project can be found here: http://www.pengutronix.de/oselas/toolchain/index_de.html

Board Support Package This is an optional component, mostly shipped aside with a piece of hardware. There are various BSP available, some are generic, some are intended for a specific hardware.

2.2.2 Extracting the Sources

To install PTXdist, at least two archives have to be extracted:

ptxdist-1.0.0.tgz The PTXdist software itself.

ptxdist-1.0.0-patches.tgz All patches against upstream software packets (known as the 'patch repository').

ptxdist-1.0.0-projects.tgz Generic projects (optional), can be used as a starting point for self-built projects.

The PTXdist and patches packets have to be extracted into some temporary directory in order to be built before the installation, for example the `local/` directory in the user's home. If this directory does not exist, we have to create it and change into it:

```
~$ cd
~$ mkdir local
~$ cd local
```

Next steps are to extract the archives:

```
~/local$ tar -zxf ptxdist-1.0.0.tgz
~/local$ tar -zxf ptxdist-1.0.0-patches.tgz
```

and if required the generic projects:

```
~/local$ tar -zxf ptxdist-1.0.0-projects.tgz
```

If everything goes well, we now have a PTXdist-1.0.0 directory, so we can change into it:

```
~/local$ cd ptxdist-1.0.0
~/local/ptxdist-1.0.0$ ls -l

total 421
drwxr-xr-x   2 jbe users   1024 2007-05-05 09:27 autoconf/
-rwxr-xr-x   1 jbe users     28 2007-05-05 09:26 autogen.sh*
drwxr-xr-x   2 jbe users   1024 2007-05-05 09:27 bin/
-rw-r--r--   1 jbe users 109082 2007-05-05 09:26 ChangeLog
drwxr-xr-x   8 jbe users   1024 2007-05-05 09:27 config/
-rwxr-xr-x   1 jbe users 206643 2007-05-05 09:26 configure*
-rw-r--r--   1 jbe users  10398 2007-05-05 09:26 configure.ac
-rw-r--r--   1 jbe users  18361 2007-05-05 09:26 COPYING
-rw-r--r--   1 jbe users   2792 2007-05-05 09:26 CREDITS
drwxr-xr-x   2 jbe users   1024 2007-05-05 09:27 debian/
drwxr-xr-x   2 jbe users   1024 2007-05-05 09:27 Documentation/
drwxr-xr-x   7 jbe users   1024 2007-05-05 09:27 generic/
-rw-r--r--   1 jbe users     58 2007-05-05 09:26 INSTALL
-rw-r--r--   1 jbe users   2686 2007-05-05 09:26 Makefile.in
drwxr-xr-x 130 jbe users   4096 2007-05-05 09:27 patches/
drwxr-xr-x   5 jbe users   1024 2007-05-05 09:37 projects/
-rw-r--r--   1 jbe users   3893 2007-05-05 09:26 README
-rw-r--r--   1 jbe users    691 2007-05-05 09:26 REVISION_POLICY
drwxr-xr-x   6 jbe users  24576 2007-05-05 09:27 rules/
drwxr-xr-x   6 jbe users   1024 2007-05-05 09:27 scripts/
drwxr-xr-x   2 jbe users   1024 2007-05-05 09:27 tests/
-rw-r--r--   1 jbe users  28468 2007-05-05 09:26 TODO
```

2.2.3 Prerequisites

Before PTXdist can be installed it has to be checked if all necessary programs are installed on the development host. The configure script will stop if it discovers that something is missing.

The PTXdist installation is based on GNU autotools, so the first thing to be done now is to configure the packet:

```
~/local/ptxdist-1.0.0$ ./configure
```

This will check your system for required components PTXdist relies on. If all required components are found the output ends with:

```
[...]
configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in

ptxdist version 1.0.0 configured.
Using '/usr/local' for installation prefix.

Report bugs to ptxdist@pengutronix.de

~/local/ptxdist-1.0.0$
```

Without further arguments PTXdist is configured to be installed into `/usr/local`, which is the standard location for user installed programs. To change the installation path to anything non-standard, we use the `--prefix` argument to the `configure` script. The `--help` option offers more information about what else can be changed for the installation process.

The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if an old version of PTXdist is already installed there is no need to remove it.

One of the most important tasks for the `configure` script is to find out if all the programs PTXdist depends on are already present on the development host. The script will stop with an error message in case something is missing. If this happens, the missing tools have to be installed from the distribution befor re-running the `configure` script.

In this early PTXdist version not all tests are implemented in the `configure` script yet. So if something goes wrong or you don't understand some error messages send a mail to `support@pengutronix.de` and help us improve the tool.

When the `configure` script is finished successfully, we can now run

```
~/local/ptxdist-1.0.0$ make
```

All program parts are being compiled, and if there are no errors we can now install PTXdist into it's final location. In order to write to `/usr/local`, this step has to be performed as root:

```
~/local/ptxdist-1.0.0$ su
[enter root password]
/home/username/local/ptxdist-1.0.0$ make install
[...]
```

If we don't have root access to the machine it is also possible to install into some other directory with the `--prefix` option. We need to take care that the `bin/` directory below the new installation dir is added to our `$PATH` environment variable (for example by exporting it in `~/.bashrc`).

The installation is now done, so the temporary folder may now be removed:

```
~/local/ptxdist-1.0.0$ cd
~$ rm -fr local/ptxdist-1.0.0
```

2.2.4 Configuring PTXdist

When using PTXdist for the first time, some setup properties have to be configured. Two settings are the most important ones: Where to store the source packages and if a proxy must be used to gain access to the world wide web.

Run PTXdist's setup:

```
~$ ptxdist setup
```

Due to PTXdist is working with sources only, it needs various source archives from the world wide web. If these archives are not present on our host, PTXdist starts the `wget` command to download them on demand.

2.2.4.1 Proxy Setup

To do so, an internet access is required. If this access is managed by a proxy `wget` command must be adviced to use it. PTXdist can be configured to advice the `wget` command automatically: Navigate to entry *Proxies* and enter the required addresses and ports to access the proxy in the form:

```
<protocol>://<address>:<port>
```

2.2.4.2 Source Archive Location

Whenever PTXdist downloads source archives it stores it project locally. If we are working with more than one project, every project would download its own required archives. To share all source archives between all projects PTXdist can be configured to use only one archive directory for all projects it handles: Navigate to menu entry *Source Directory* and enter the path to the directory where PTXdist should store archives to share between projects.

2.2.4.3 Generic Project Location

If we already installed the generic projects we should also configure PTXdist to know this location. If we already did so, we can use the command `ptxdist projects` to get a list of available projects and `ptxdist clone` to get a local working copy of a shared generic project.

Navigate to menu entry *Project Searchpath* and enter the path to projects that can be used in such a way. Here we can configure more than one path, each part can be delemited by a colon. For example for PTXdist's generic projects and our own previous projects like this:

```
/usr/local/lib/ptxdist-1.0.0/projects:/office/my_projects/ptxdist
```

Leave the menu and store the configuration. PTXdist is now ready for use.

2.3 Toolchains

2.3.1 Abstract

Before we can start building our first userland we need a cross toolchain. On Linux, toolchains are no monolithic beasts. Most parts of what we need to cross compile code for the embedded target comes from the *GNU Compiler Collection*, `gcc`. The gcc packet includes the compiler frontend, `gcc`, plus several backend tools (cc1, g++, ld etc.) which actually perform the different stages of the compile process. `gcc` does not contain the assembler, so we also need the *GNU Binutils package* which provides lowlevel stuff.

Cross compilers and tools are usually named like the corresponding host tool, but with a prefix – the *GNU target*. For example, the cross compilers for ARM and powerpc may look like

- `arm-softfloat-linux-gnu-gcc`
- `powerpc-unknown-linux-gnu-gcc`

With these compiler frontends we can convert e.g. a C program into binary code for specific machines. So for example if a C program is to be compiled natively, it works like this:

```
~$ gcc test.c -o test
```

To build the same binary for the ARM architecture we have to use the cross compiler instead of the native one:

```
~$ arm-softfloat-linux-gnu-gcc test.c -o test
```

Also part of what we consider to be the ”toolchain” is the runtime library (libc, dynamic linker). All programs running on the embedded system are linked against the libc, which also offers the interface from user space functions to the kernel.

The compiler and libc are very tightly coupled components: the second stage compiler, which is used to build normal user space code, is being built against the libc itself. For example, if the target does not contain a hardware floating point unit, but the toolchain generates floating point code, it will fail. This is also the case when the toolchain builds code for i686 CPUs, whereas the target is i586.

So in order to make things working consistently it is necessary that the runtime libc is identical with the libc the compiler was built against.

PTXdist doesn't contain a pre-built binary toolchain. Remember that it's not a distribution but a development tool. But it can be used to build a toolchain for our target. Building the toolchain usually has only to be done once. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

2.3.2 Using Existing Toolchains

If a toolchain is already installed which is known to be working, the toolchain building step with PTXdist may be omitted.

The OSELAS.BoardSupport() Packages shipped for PTXdist have been tested with the OSELAS.Toolchains() built with the same PTXdist version. So if an external toolchain is being used which isn't known to be stable, a target may fail. Note that not all compiler versions and combinations work properly in a cross environment.

Every OSELAS.BoardSupport() Package checks for its OSELAS.Toolchain it's tested against, so using a different toolchain vendor requires an additional step:

Open the OSELAS.BoardSupport() Package menu with:

```
~$ ptxdist menuconfig
```

and navigate to `PTXdist Config, Architecture` and `Check for specific toolchain vendor`. Clear this entry to disable the toolchain vendor check.

2.3.3 Building a Toolchain

PTXdist-1.0.0 handles toolchain building as a simple project, like all other projects, too. So we can download the OSELAS.Toolchain bundle and build the required toolchain for the OSELAS.BoardSupport() Package.

A PTXdist project generally allows to build into some project defined directory; all OSELAS.Toolchain projects that come with PTXdist are configured to use the standard installation paths mentioned below.

All OSELAS.Toolchain projects install their result into `/opt/OSELAS.Toolchain-1.1.0/`.

Usually the `/opt` directory is not world writable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions so that the user can write (`mkdir /opt/OSELAS.Toolchain-1.1.0 ; chown <username> /opt/OSELAS.Toolchain-1.1.0; chmod a+rwx /opt/OSELAS.Toolchain-1.1.0`).

2.3.3.1 Building the OSELAS.Toolchain for OSELAS.BSP-JB-GeoTerm-2

To compile and install an OSELAS.Toolchain we have to extract the OSELAS.Toolchain archive, change into the new folder, configure the compiler in question and start the build.

The required compiler to build the OSELAS.BSP-JB-GeoTerm-2 board support package is

`i586-pmmx-linux-gnu_gcc-4.1.2_glibc-2.5_linux-2.6.18`.

So the steps to build this toolchain are:

```
~$ tar xf OSELAS.Toolchain-1.1.0.tgz
~$ cd OSELAS.Toolchain-1.1.0
~/OSELAS.Toolchain-1.1.0$ ptxdist select
  ptxconfigs/i586-pmmx-linux-gnu_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig
~/OSELAS.Toolchain-1.1.0$ ptxdist go
```

At this stage we have to go to our boss and tell him that it's probably time to go home for the day. Even on reasonably fast machines the time to build an OSELAS.Toolchain is something like around 30 minutes up to a few hours.

Measured times on different machines:

- Single Pentium 2.5 GHz, 2 GiB RAM: about 2 hours
- Dual Athlon 2.1 GHz, 2 GiB RAM: about 1 hour 20 minutes
- Dual Quad-Core-Pentium 1.8 GHz, 8 GiB RAM: about 25 minutes

Another possibility is to read the next chapters of this manual, to find out how to start a new project.

When the OSELAS.Toolchain project build is finished, PTXdist is ready for prime time and we can continue with our first project.

2.3.4 Freezing the Toolchain

As we build and install this toolchain with regular user rights we should modify the permissions as a last step to avoid any later manipulation. To do so we could set all toolchain files to read only or changing recursivley the owner of the whole installation to user root.

This is an important step for reliability. Do not omit it!

2.3.4.1 Building additional Toolchains

The OSELAS.Toolchain-1.1.0 bundle comes with various predefined toolchains. Refer the `ptxconfigs/` folder for other definitions. To build additional toolchains we only have to clean our current toolchain projekt, removing the current `ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.1.0$ ptxdist clean
~/OSELAS.Toolchain-1.1.0$ rm ptxconfig
~/OSELAS.Toolchain-1.1.0$ ptxdist select
       ptxconfigs/any_another_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.1.0$ ptxdist go
```

All toolchains will be installed side by side architecture dependend into directory

`/opt/OSELAS.Toolchain-1.1.0/architecture_part`.

Different toolchains for the same architecture will be installed side by side version dependend into directory

`/opt/OSELAS.Toolchain-1.1.0/architecture_part/version_part`.

3 Building Description

3.1 Prepare and Build

This chapter assumes we installed ptxdist and the toolchain in a way the last chapter describes it.

It also assumes we have a big wire into the internet, as all the sources are downloaded from there. If we setup ptxdist to use a generic source archive directory the download only happens once.

Each ptxdist project uses a toolchain. This is the first thing we must setup prior starting the build.

```
jb@jupiter:~/ $ cd OSELAS.BSP-JB-GeoTerm-2
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist toolchain
            /opt/OSELAS-1.1.0/i586-pmmx-linux-gnu/gcc-4.1.2-glibc-2.5/bin
```

Now everything is ready to build the project:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist go
```

This will take a while. The X packet is large and needs the most time to build.

Sometimes downloading a source archive may fail. Most of the time the maintainer moves the archive to another location or some delete the older releases when a new version was released. A simple solution is to search for this archive in the web, download it manually and store it into the generic source archive directory. If the archive file is already present, no download will happen.

After building the project, we can find everything we need in the `root/` directory. This directory is intended to be used as an NFS root filesystem. To do so, we must fill the `root/dev/` directory with at least three device nodes: `console`, `null` and `zero`. ptxdist rejects to work as root, so we must create these nodes manually. We could do so with:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ sudo cp -a /dev/console /dev/null
            /dev/zero root/dev
```

[*sound of entering root password*]

Done. Now we could boot this filesystem as our IGEL's root filesystem.

But first we must replace the IGEL's current BIOS by LinuxBIOS. I found no other way then using an external flash programmer to replace flash's content.

Refer http://www.linuxbios.org/index.php/BCOM_WINNET100_Build_Tutorial how to get access to the flash device. The LinuxBIOS file to flash we can found in `images/linuxbios.rom`.

3.2 Environment Setup

To use this modified system, we need a DHCP service. Etherboot as LinuxBIOS's payload sends a DHCP request and waits for the answer what kernelfile it should load. This implies also an running TFTP service. The kernel file to load can be found in one of the `images/*.kernel` files.

The same is valid for the kernel: It also uses the DHCP service to get its NFS root filesystem path.

3.3 Configuration

If anything does not work, we need some configurations changes, as the default configuration meets my small home net only.

3.3.1 Kernel Command Line

To change kernel parameter embedded in the the kernel files, we start the configuration menu:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist menuconfig
```

navigate to **Project Specific Configuration** and **Build etherboot kernels** and change the **Kernel base line** or any of the system specific line extensions to meet our local requirements.

3.3.2 Userland Configuration

If the userland crashes there are several places where changes are possible. It depends on the kind of crash, what file or setting must be modified.

Here is a list of possible location where to change something:

- I'm using a font server. This setting is done in **Project Specific Configuration**, **install xorg.conf**. Enter here the correct name of your server.
- If you need a special fstab setting, modify `projectroot/etc/fstab`
- If you need a special `group|gshadow|passwd|shadow-` setting, modify `projectroot/etc/group|gschadow|passwd|shadow-`
- If you need a special `inittab` setting, modify `projectroot/etc/inittab`
- If you need a special `modules` setting, modify `projectroot/etc/modules`
- If you need a special `nsswitch.conf|resolv.conf` setting, modify `projectroot/etc/nsswitch.conf|resolv.conf`
- If you need a special `xorg.conf` setting, modify `projectroot/etc/xorg.conf`

After changing anything in one of this files, we must inform ptxdist about this change and let it create a new root filesystem in sync with the changes:

```
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist drop rootfs
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist drop fixup-rootfs
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist go
```

3.3.3 LinuxBIOS Configuration

This BSP contains a few configuration options to build LinuxBIOS. We can select if the LinuxBIOS should activate the VGA subsystem or not and if it should do so, we can select what screen resolution it should use.

Also some optimised memory configurations are available (very slow timing, optimised timings for several memory types).

To setup these things we start the menuconfigurator

```
jb@jupiter:~/OSELAS.BSP-JB-GeoTerm-2 $ ptxdist menuconfig
```

navigate to **Project Specific Configuration** - **Build LinuxBIOS** and select one of the possible options in **Build type**. If we select one of the active VGA type we can also select a boot screen resolution in **Screen resolution**.

3.4 Changing the Boot Splash Graphic

If you don't like my boot splash graphic, simply change the XPM graphic file in `local_src/xpm-converter/input.xpm`. Keep it small, as it must fit into the flash ROM. The colour of the left top pixel will be used for the whole screen, and the image will be displayed at screen's right bottom.

4 System Description

This chapter describes some details of the system.

4.1 The Geode GX1 specific Drivers

4.1.1 System Management Replacement

Original Geode GX1 based systems using the *System Management Mode* (=SMM) extensivley. Almost everything is controlled by this piece of software called VSA. Video, Audio, Power Management and so on. But the SMM is also the reason for many frustration as it (like all software) contains errors and disturbes some system activity (realtime application for example).

4.1.1.1 Notes about the hardware

The chipset and the CPU itself supports many checkpoints where some activity will generate a *System Management Interrupt* (=SMI). This will activate the VSA code, that checks what to do with this activity. One activity could be the access to any of the VGA compatible control registers. These registers do not exists on Geode GX1 hardware, so the VSA software emulates them. This let the Geode system acts like a full compatible PC, without a full compatible PC hardware.

Yes, its a nice idea. But: When you access one of the VGA control registers, this takes on old ISA hardware about 1 microsend. On modern PCI based hardware it is much faster, lets say about 66 nanoseconds. In contrast the emulated access needs several dozen microseconds (or maybe 100 and more).

Most of the features the SMM hardware supports we do not need unless we want to get our system into some kind of sleep mode. Only one hardware part relys on the SMM and SMI: The sound. Its a PCI master DMA capabilty hardware that wakes the CPU with an SMI when the current data block was sent to the AC97 codec. Currently there is no way to forward this SMI to a regular interrupt.

In my BSP I support this SMI by a small driver that polls the SMI status registers and if there is some activity, it calls a previous registered callback function. Polling is ugly, I know. But I have no other idea yet.

The current sound driver uses this polling feature to forward the next sound packets to the DMA engine.

4.1.2 Framebuffer Driver

The framebuffer driver uses Geode's native hardware to support a system console. It uses the acceleration features of this chipset to speed up text drawing and some special features for high resolution screen.

To configure the frambuffer driver we can add this to the kernel command line:

video=gx1fb:<params>

<params> could be:

- **monitor** this defines the connected monitor and its supported resolution and features.
  This parameter needs x and y resolution and frame refresh rate. An XGA TFT monitor could be defined with **monitor:1024x768@60**, an SXGA TFT monitor with **monitor:1280x1024@60**
- **mode** this defines the graphics mode, colour depth and frame refresh rate to be used for the console.
  If not defined the native monitor resolution with 256 colours will be used. With this parameter a smaller mode can be defined. To use a VGA resolution with 64k colours and 60Hz refresh rate for the console we can use **mode:640x480-16@60**.

There is a special mode support for generic XGA and SXGA monitors: It uses the native monitor resolution but only the half resolution in memory by displaying each pixel and each line twice. For an SXGA monitor this could be configured with: **mode:640x512-8@60,bigchar:1**.

- **crt** switches on or off the CRT (analouge) output. **crt:1** switches analouge output on, **crt:0** switches it off.
- **panel** switches on or off the TFT (digital) output. Untested! I have no hardware to test.
- **bigchar** Enable pixel and line doubling for high resolution modes. This could save power and memory bandwith as it halfes the pixels the chipset must handle.

Some examples:

Connected is an XGA TFT monitor that likes 60Hz refresh rate. We want use it with a 256 colour full resolution console:

video=gx1fb:monitor:1024x768@60,crt:1

Connected is an SXGA TFT monitor that likes 60Hz refresh rate. We want use it with a 256 colour quarter sized resolution console:

video=gx1fb:monitor:1280x1024@60,mode:640x512-8@60,crt:1,bigchar:1

Connected is an SXGA TFT monitor that likes 60Hz refresh rate. We want use it with a 64k colour full resolution console:

video=gx1fb:monitor:1280x1024@60,mode:1280x1024-16@60,crt:1

4.1.3 Xorg server

4.1.3.1 Notes about the video hardware

The Geode processor and its companion chip supports screen resolutions up to 1280x1024 with up to 16 bit depht. 1280x1024 at 16 bit is not easy to handle because the device supports only 4MiB of shared video memory. This sounds confusing, as SXGA with this pixel depht only require 1280 * 1024 * 2 = 2560kiB.

But the device cannot address the lines continous in memory when acceleration and/or compression is in use. They always requires a fixed offset (e.g. delta) per line to get the start position of the next line. This delta can be 1024, 2048 or 4096 bytes. As of a 1280 pixel line with 16 bit depth requires 2560 bytes only the 4096 bytes offset per line can be used. And 1024 lines with 4096 bytes each consumes the full 4MiB video memory.

Note: If wether compresson nor acceleration is used, there is no such restriction for the line delta value. Every line must start at a DWORD boundary only.

There are a few bits in register GP_BLT_STATUS that must be set in accordance to the line delta. They are located in the acceleration hardware, but configure the compression hardware. Blit operations are independend from their settings. There must be another setting, that tells the blit engine where it finds the next line start offset in the framebuffer. I think it uses bits [10..8] of register DC_LINE_DELTA. It seems the highest bit wins, as only 1024, 2048 or 4096 offsets are working correctly. Settings like 3072 don't work.

The acceleration hardware needs a temporarely pixel buffer. It is located in the so called *Scratch Pad RAM*. This is a part of the CPU cache and must be reserved from the use as CPU cache before. On a VMA BIOS this is fixed when the system boots. But if there is no VMA, there is no need to reserve a fixed amount of scratch pad RAM from the cache. Reservation is handled by my framebuffer console driver in collaborating with my chipset driver. So the scratch pad RAM size can depend on screen resolution instead of a fixed value. And: The whole reserved memory can used. With a VMA some of this space is occupied by the VMA itself.

My Xorg driver requests for 2048 bytes for all resolutions up to XGA and for 4096 bytes for SXGA. It could be possible to ask for 3072 bytes only in SXGA. But it seems there is a defect in the hardware, as with this setting blit operations wider than 1024 pixel fail. I don't know why.

The acceleration hardware has some nice monochrome pattern functions. With this its very easy and fast to render letters into the framebuffer. This feature will be used in my framebuffer console driver. The frambuffer console framework will deliver the monochrome characters, background and foreground colour and the rest is done independendly from colour depth in hardware.

Mostly used to emulate one of the old and ugly videomodes Geode's hardware can double pixels and lines. With this feature my framebuffer console driver can handle large screens (lets say 1280x1024) with a smaller console (640x512). But the connected flatscreen sees a SXGA timing, while the video hardware generates lower rated data. It saves memory bandwidth and enlarges the letters on the screen (but keep them sharp as flatscreen's native resolution is in use).

4.1.3.2 Xorg Video Driver for Geode GX1 based systems without VMA

This Xorg driver depends on the Geode Accelerated Framebuffer Console driver. Only the bitblit operations are handled by the Xorg driver. To handle the display, resolution and colour depth the framebuffer console driver will be called via IOCTL. Why to copy and paste code, when the framebuffer console driver can deliver most of the requested features?

This driver is only tested with XGA and SXGA resolutions. The flatscreens you can buy today have SXGA resolution. My XGA testscreen is a very old one. So the best tested environment is 1280x1024-16@60Hz. This means: SXGA resolution, 64k colours and 60Hz refresh rate.

As the framebuffer console driver controls all the timings and resolutions settings the Xorg driver is responsible to organise the framebuffer in a way to use all the hardware features the Geode supports.

Things that need the video frame memory:

- Visible Frame Buffer: It must start always at offset 0 of the frame buffer memory to also use the compression feature. Also the line delta is an important value that take effect on the other features.
- Compressed Frame Buffer: It contains compressed line data from the visible part to save memory bandwidth.
- Cursor Image Data: To use the hardware cursor, 256 bytes must be reserverd to store the image data.
- Blit sources (images, rasters and so on)

To organise these data sets there are only two ways. Mostly they rely on the line delta. Figure 4.1 shows the two possible ways.

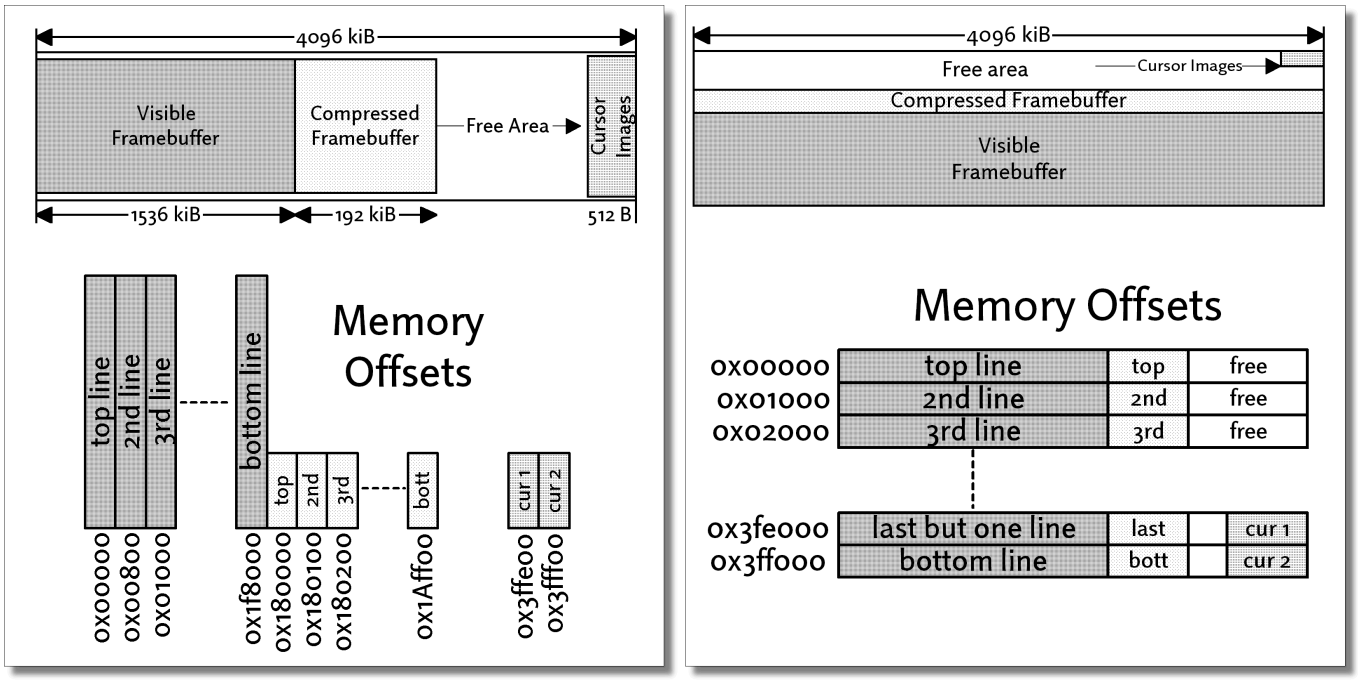

Figure 4.1: *Left: XGA layout, right: SXGA layout*

XGA video memory layout

Refer to the left side at figure 4.1. XGA at 64k colours needs a line delta of 2048 bytes. This is one of the possible line delta settings so the visible part of the framebuffer is continuous. After this part follows the compressed buffer. Each compressed line can only contain up to 256 bytes of data. If the hardware can't deflate the visible line to this limit it always display the uncompressed line content (with higher bandwidth). If it fits into the 256 bytes, it uses the compressed data, until someone writes into the corresponding uncompressed line. Then compression starts again one time and so on. The 256 bytes are not a fixed value but it is the maximum. For smaller line sizes also a smaller compressed line size would be suitable. But for 1024 or 1280 pixels per visible line, many lines could not deflate down to 256 bytes. So the advantage of this feature in lower memory bandwidth and lower power consumption is only limited. But for large unicoloured areas it works.
At the end of the whole video memory the cursor images are located. I reseve two buffers to avoid cursor image disturbing while loading a new image. Loading always occures into the currently invisible buffer. After loading it switches over to this buffer.
Currently there is no support to store anything into the free area (in X notation it is called *offscreen memory*) for acceleration.

SXGA video memory layout

Refer to the right side at figure 4.1. SXGA at 64k colours needs a line delta of 4096 bytes. This results in the whole allocation of the 4MiB video memory, but with many unused bytes between each line. 1280 pixels at 64k colours need 2560 bytes. So there remains 1536 unused bytes per line.
These unused bytes are used for the compression buffer and for the cursor images. Each line of the compressed buffer is located behind the visible uncompressed line.
The cursor images are located between the last two lines in the whole video memory each at the end of the 1536 bytes area. So they do not conflict with the compressed data buffer.
Enabling compression makes sense at SXGA resolution. SXGA with 64k colours and 60Hz refresh rate consumes about 153.6 MiB per second bandwidth into the memory. This bandwidth will be stolen from the CPU and the acceleration hardware, so compression also improves overall speed.

4.1.3.3 Setting up PTXdist to get a working X server

In this BSP the Geode GX1 native Xorg driver is part of the BSP, no external package. Nevertheless the Xorg video drivers must be activated in the menu, but no real driver in this menu is required. The Geode GX1 native Xorg driver will be activated at the toplevel of the menu at *Install Geode framebuffer X driver*. To make the whole Xorg running on the target also *Support keyboard mappings* in the *overall X11 options* must be enabled.

4.1.3.4 Configuring the X server

Everything to configure the runtime Xorg we will find in */etc/xorg.conf*. Most of the settings are generic, but some are specific to the native Geode GX1 driver.

```
Section "Device"
        Identifier      "geode_native"
        Driver          "fbdev"
        BusID           "PCI:0:18:4"
        Option          "ShadowFB" "0"
        Option          "Accel" "1"
        Option          "Compression" "1"
# our CPU lacks mtrr support
        Option          "MTRR" "0"
        Option          "fbdev" "/dev/fb0"
EndSection
```

Currently its name is still fbdev (from the framebuffer driver I start from to develop this driver). This will be changed in the near future.

BusID is a hint to the driver, what PCI device the VGA is. *ShadowFB* is currently not supported (most other driver will switch off any hardware acceleration when the shadow frame buffer is enabled. Don't know why), keep it disabled would be the best value for the future. *Compression* is currently only supported for SXGA modes. But it is worth to be enabled. *Accel* will control the use of the hardware acceleration. If the videomemory is very small maybe a big

linedelta would waste to much memory. With acceleration off the linedelta is only 0..3 bytes per line. *MTRR* should be disabled on the Geode GX1, as this processor does not support such hardware. *fbdev* is a hint to the driver, what framebuffer device node it should use.

To use the driver create a monitor and a screen section that refer to Geode's device section:

```
Section "Monitor"
        Identifier      "mona_monitor"
        VendorName      "ACER"
        ModelName       "AL1716"
        HorizSync       47-65
        VertRefresh     59-61
        DisplaySize     340 270
        Option          "TargetRefresh" "60.0"
        Option          "dpms" "1"
EndSection

Section "Screen"
        Identifier      "mona_screen"
        Device          "fbdev"
        Monitor         "mona_monitor"
        DefaultDepth    16

        SubSection "Display"
                Depth           16
                Viewport        0       0
                Virtual         1280    1024
                Modes           "1280x1024"
        EndSubSection
EndSection
```

4.1.4 Sound on the Geode

With a regular BIOS the Geode GX1 based systems support a SoundBlaster compatible soundcard. This sounds good, but the disadvantage is, to play sounds it consumes up to 50% of the CPU. Because its only emulated.

4.1.4.1 Notes about the hardware

The native Geode hardware is designed to interact with the system management mode. Due to this the first restriction to use it without the SMM is the lack of interrupt generation. The native sound hardware can only generate a system management interrupt. I found no way to forward it to a regular interrupt.

The native sound hardware supports AC97 and master DMA to transfer the sound data in both directions (record and play). To handle each end of DMA transfer an interrupt is required to inform the framework. Due to the lack of an interrupt only polling the status bits is possible. Sorry about that, I found no other solution yet. The good news is, that polling is disabled, when there is no need for it (no sound is playing). Its done only on demand (saves power).

4.1.4.2 Geode ALSA Sound Driver

The native ALSA driver relies on the small SMI polling driver to get something similar to an interrupt. Whenever the hardware tries to generate an SMI, the SMI polling driver calls sound driver's callback function.

Currently only playback can be used with this driver.

4.1.4.3 Configuring Sound at Runtime

We can use the regular ALSA tools to configure the sound driver. A small startup script restores at system start the last saved mixer settings. To save the mixer setting we must run ”alsactrl store 0” one times after we have configured the mixer itself (run ”alsamixer”).