

OM-Cube project

V. Hiribarren, N. Marchand, N. Talfer, P Kadionik

Abstract. The OM-Cube project aims to build an embedded multimedia player using open source software. It will process multimedia streams on a television screen. A first working Linux distribution has been built but does not feature a TV display and IR remote control yet. In compensation the current system supports a LCD display and a distribution generator has been created to ease configuration tests. The current hardware delivers poor performances. Several problems have been encountered due to the compact flash card, compilation time and incompatibilities between the kernel configuration and the hardware.

Introduction

The OM-Cube is a compact Open MultiMedia Machine with IR remote control based on Linux which displays different kinds of multimedia streams on a TV screen: audio, video and photo. The OM-Cube player must be small so as to be used in a living-room and easy-to-use for anybody. Furthermore, lots of open source software is designed to work on common computers. Therefore a small dedicated personal computer will be used.

Some similar projects already exist but building its own embedded multimedia player enables a better control. It is also a pedagogical way to discover the operation of a Linux system and the embedded world.

The system will use a Linux kernel for several reasons. First, Linux is stable, open source and requires low memory. Then it will enable us to use open source software based on a unix kernel and to implement applications in a high level language. The embedded operating system must require low memory to fit the target compact flash memory size. The processor must also be powerful enough to process video streams. Then, several open source players have to be tested (MPlayer, Xine...) in order to choose the most efficient one. The last step will be the integration of hardware functionalities such as a LCD screen or an IR remote control.

After having linked the project to the bibliographical study, each layer of the embedded system is going to be discussed: hardware, operating system and user land applications. Finally the encountered problems and the remaining work will be dealt with.

1 Related Work

A previous bibliographical study [1] introduced several concepts supposed to be tackled during the project. Some of them have been very helpful whereas some others have been forgotten.

Hardware. Because of a limited budget, it has not been possible to get a more efficient hardware. The CPU does not deliver acceptable performances to play video streams. Neither TV-OUT connector nor infrared captor was available on the motherboard and it would have been too difficult to integrate ones. Instead of working on a complete system, the project focused on a basic working system with tests on performances. Nevertheless, the LCD display has been successfully implemented.

Operating System Selection. Only GNU tools and the Linux kernel have been tested. All other considered choices have been discarded due to a lack of time. On another hand, a new tool which was not discussed until now has been developed in order to automatically generate a Linux distribution.

Graphic Layer. Four graphic layers were discussed: XFree86, framebuffer, VESA and DirectFB. DirectFB has not been tested because too many software do not support it. Concerning XFree86, it is confirmed it requires too many resources. Work has been concentrated on VESA and framebuffer. It is reminded that VESA is a graphic card BIOS which allows to drive directly the display. Framebuffer is a unix abstraction which provides a display management through the kernel and can use either VESA or a specific driver.

Multimedia Tools. The open source video jukebox Freevo has been tested; it is based on a number of open-source audio/video tools. However this tool needs a Python interpreter and related libraries which consume too much memory for an embedded system. No other front-end has been experimented: time was missing. The choice of the multimedia player was only made between MPlayer and Xine.

2 Hardware Description

2.1 The Barebone

The barebone contains the same components as a common PC (memory, processor, network board, graphics card, motherboard), with an external DVD-player, and a compact flash card (128 Mbytes) to store software, drivers and the operating system. The processor is a Cyrix III clocked at 533 Mhz with 128 Mbytes of SDRAM memory and a trident CyberBlade chip-set graphic card (8 Mbytes). Even if these resources seem to be limited, they are quite common in an embedded system.

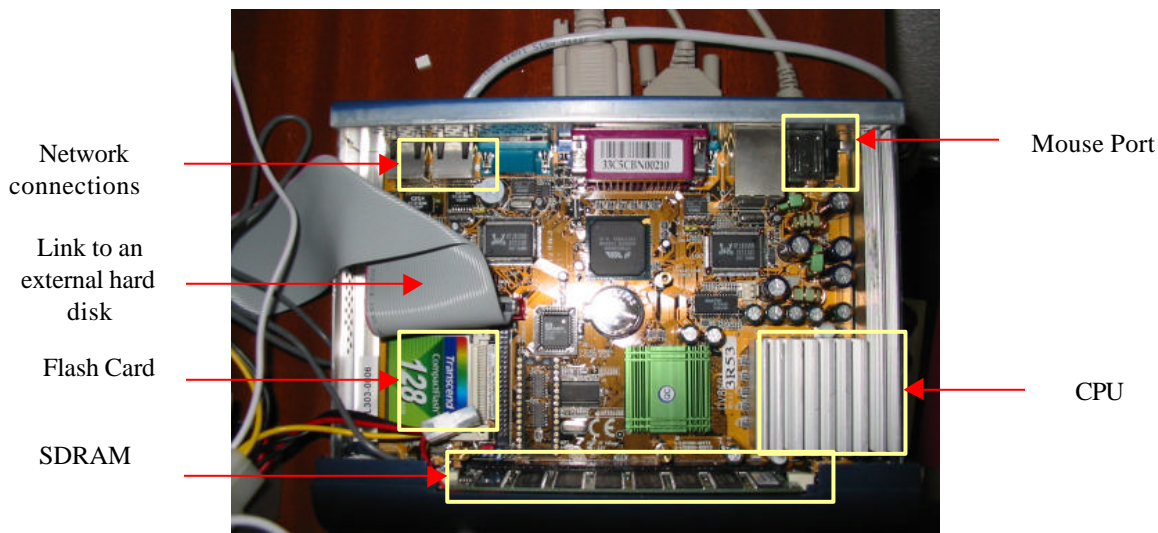


Figure 1: inside the barebone.

The memory size of the flash card is sufficient to store a minimal operating system with well chosen user land applications. Nevertheless, more space is needed to build, configure and test this system. That is why a hard disk is used to develop the system before transferring it on the flash card. This external hard disk contains a RedHat Linux distribution.



Figure 2: the barebone and other devices.

The OM-CUBE project needs a television output (via an S-Video output for instance) in order to display video streams on a TV screen. But the current motherboard does not have a TV-OUT connector. Moreover, the new motherboard that will be used to continue the project by other students will have such a connector and better video performances.

2.2 The LCD Device

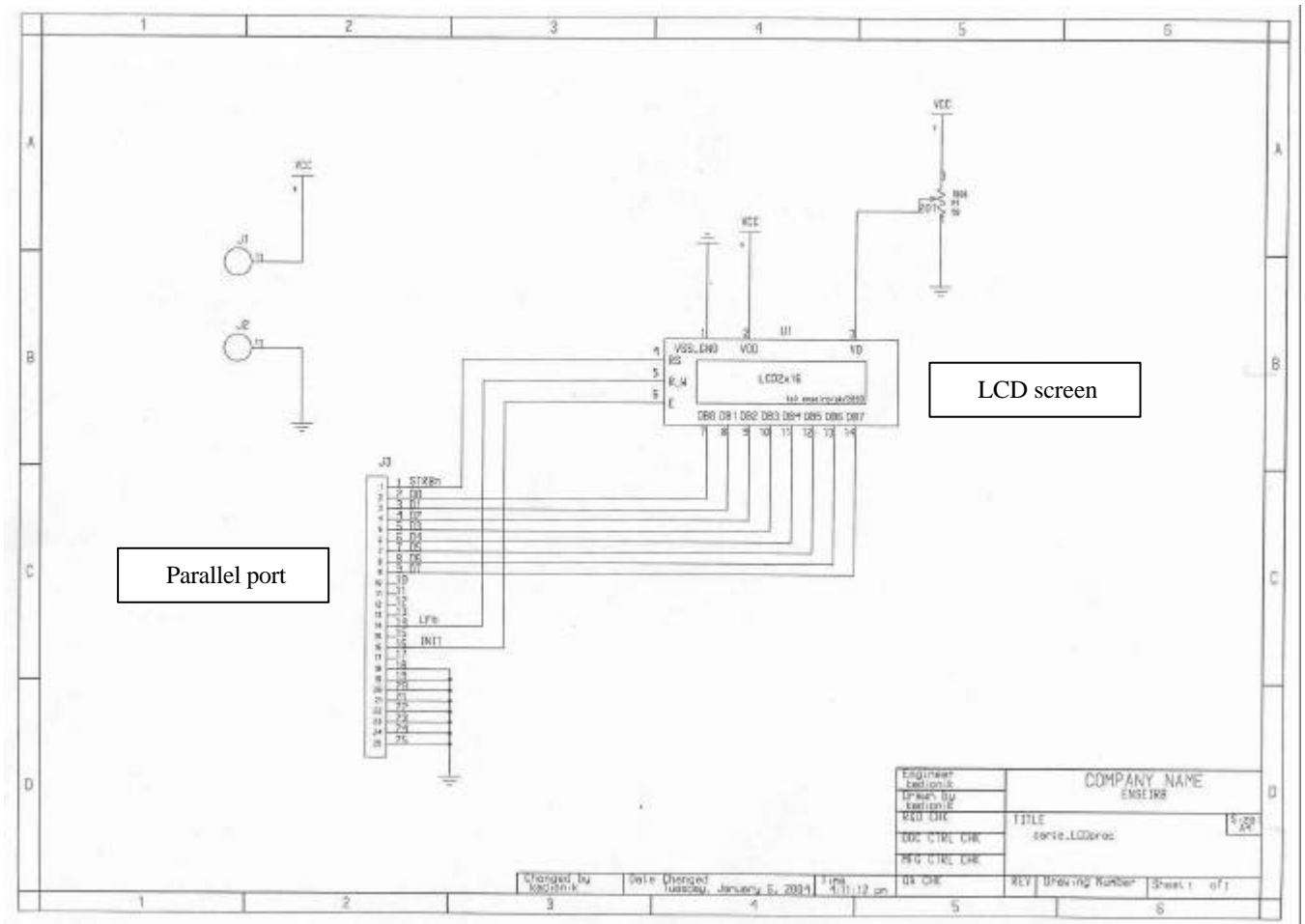


Figure 3: connection scheme of a LCD device to a parallel port.

A LCD screen is used to display information about the current multimedia stream (i.e the current time, the current chapter for DVD, the song title for MP3...). The LCDd daemon taken from the lcdproc project provides a software interface to drive the LCD display. Thanks to the project tutor and some schemes found on the Internet, a LCD screen has been made and is used through the parallel port of the barebone.

3 The Operating System

3.1 Building of a Minimal Linux Distribution

One goal of this project is the construction of a minimal Linux distribution from scratch. It is a pedagogical way to understand the operation of a GNU/Linux system. But it also allows building an operating system adapted and tuned to the hardware and the project needs.

What Is Needed. Four components are needed to build our GNU/Linux operating system:

- a boot loader, which will be LILO
- a Linux kernel, patched and well configured with the framebuffer support.
- a C-library which contains basic functions used by software: glibc or μ Clibc for the current system
- system utilities, like *init*, *mount* and a shell to read script files

These main system utilities can easily be replaced by the use of Busybox which combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most of the utilities like fileutils, shellutils, findutils, textutils, grep, gzip, tar, etc. BusyBox provides a fairly complete POSIX environment for any small or embedded system. BusyBox has been written with size-optimization and limited resources in mind.

In order to display a nice logo instead of the ugly common characters at boot time, the kernel is patched with Bootsplash: it makes the OM-Cube friendlier to the user.

It can be noticed that the *mklibs.sh* tool from the Debian project has been tested: it helps removing useless parts of libraries, but it had not been used while building our system.

A Method. The method consists in picking up elements from an existing Linux distribution: Redhat. On the host system, a temporary directory is created and filled with needed files.

The *ldd* utility enables to know which libraries an application needs. To ensure all the components are well placed, the *chroot* command is used to change the root directory to the temporary directory: programs can be tested as if they were on the flash card.

When all the components have been chosen, it is time to put them on the target system: the flash card. A primary partition is built so as an ext3 file system. No swap file system is required, as the memory will only contain useful processes and is big enough to avoid swapping.

The last step implies the installation of the boot loader on the flash card, to allow booting on it when the barebone is powered on.

3.2 Linux Distribution Generator

Description. In order to automate the construction of a Linux distribution for the OM-Cube, a distribution generator was built. This generator allows easy choosing and testing of different software configurations. It is a set of shell scripts, configuration files and software compressed packages. The aimed features are:

- using of compilation scenario to choose which software components are included.
- checks some software configuration changes to compile only what is necessary and to gain time.
- fills the destination file system with the useful files.
- patches the packages with included patches.

Apart from the auto-patcher, these features are currently reached, but too few software are available in the tool so one can only act on the software configuration files. The *mklibs.sh* tool should be used to extract automatically what is not needed in the libraries, but it is not yet implemented.

To explain how the generator processes, here is its directory structure:

- config: configuration of the generator, and compilation scenario files.
- scripts: main script files.
- packages: all the packages, in separate directories, with their configuration files and three tiny scripts which automate their compilation (*configure-default.sh*, *build-default.sh*, *install-default.sh*). Currently, only the Linux kernel, busybox, lilo, glibc, μ Clibc and MPlayer are available.
- obj: packages are decompressed and compiled here.
- final: the final file structure that has to be put on the flash card.

The main script files process and control the making of the new distribution.

Processing. First of all, a scenario file is read to know which components will be compiled. This method allows the user to easily change or add a component, like using μ Clibc instead of the glibc. At the present time, components must be put in the right order in the scenario: if a program needs the glibc, glibc must be placed before this software.

The needed packages are decompressed into the “obj” directory if it was not done in a previous use of the generator. Then, packages are scanned and three scripts tied to each package are called: one to configure the package, another to compile it and the last to install the right components in the “final” directory.

In order to avoid a reconfiguration and a recompilation of an already well compiled package, the configuration files previously used and those which will be used are compared: if there is no change, no reconfiguration will be done. The process to know which files have to be recompiled is done by the usual “make” tool which use “Makefile” included in packages.

If one wants to add a package to the generator, the compressed sources with the right configuration files must be put in the “packages” directory with three tiny scripts:

- config-default.sh: how the package must be configured and interact with the other packages.
- build-default.sh: how to start the compilation, and which parameters have to be given.
- install-default.sh: which files of this package will be taken and put in the final filesystem

Problems. While this system can quickly offer tuned distribution, a long study of the packages must be done before their integration. Each package has a different configuration or compilation method. Sometimes these methods are documented, sometimes they aren't. As the generator must automate the configuration and the compilation while not requiring the user intervention during the process, all the useful configuration files must be spotted; the configuration script must answer questions if the package's tools required some.

The major issue was to carefully take into account the dependencies between the packages: for instance, μ Clibc needs the headers of the Linux kernel which will be used in the future distribution; busybox and MPlayer have to be compiled while using the μ Clibc library. If one uses these packages for the first time, there is a real waste of time used to test different configurations before getting functional configuration files. Version packages are also quite important: μ Clibc-9.20 provides wrappers for the gcc compiler which allow using μ Clibc without any further tools. Nevertheless bad compilation can occur, and one would like to use the latest μ Clibc version... which does not provide compilation tools, but needs a complete tool-chain of programs or a whole environment designed to compile programs with μ Clibc.

Consequently if the generator distribution is a basic and simple idea, the construction of such a tool is quite long and laborious.

4 Userland Software

4.1 Choosing the Multimedia Player

Two media players have been tested: MPlayer and Xine. This software can play most of the popular multimedia file formats.

MPlayer. A great feature of MPlayer is the wide range of supported output drivers. It works with lots of graphic layers like XFree86 and the framebuffer. MPlayer has an onscreen display (OSD) for status information and visual feedback for keyboard controls.

First, MPlayer has been tested with Xfree86; performances were limited: the display on the screen was slowing down and jerky. The sound was blocked about thirty seconds with a message from MPlayer: "The system is too slow to play this file". In order to improve this result, options for better performances have been modified but the result was very close: too slow. To avoid the use of X with the hope of getting better performances, either VESA mode or framebuffer should be used. But in a first time, it has been impossible to enable framebuffer. This problem has been solved later and will be discussed further in this document.

Thus the minimal system was launched in framebuffer mode which enables a better control of the graphic display than VESA. However the display is still better when MPlayer is launched with VESA. More tests should be carried out to optimize the running of MPlayer under framebuffer.

Xine. Xine is another popular multimedia player which has been tested. The features are quite similar to MPlayer, except that Xine does not support VESA. Many forums and articles about Linux media players consider Xine as less efficient than MPlayer but easier to customize.

In fact, Xine was really slower than MPlayer with Xfree mode (about four images per second). Despite the customization, Xine did not fit with the requirements of the OM-Cube project (in any case less than MPlayer). As Xine is less efficient under framebuffer than MPlayer and does not support VESA, MPlayer seemed to be the most sensible choice to carry on the tests.

4.2 Interfacing the LCD Screen with MPlayer

In order to display information on the LCD screen, MPlayer has to share data on the multimedia streams with the LCD architecture. This architecture uses the LCDproc utility according to the following scheme.

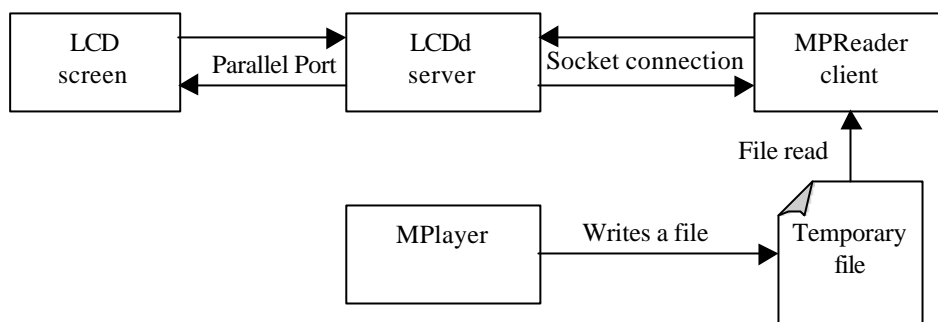


Figure 4: LCD architecture.

LCDproc uses a server and a client. The server enables to display messages which are sent from the client throughout a client/server model. In order to display information about media streams on the LCD, the main C file of MPlayer was modified. It writes temporary files in which information is gathered.

The client reads these files and sends to the LCDd server what to display: the title of a DVD and the current progress of the movie. Instead of using temporary files, alternative and simpler architectures could have been selected but they would have required more modifications in MPlayer sources.

5 Encountered Problems

During this project, we have come up against several difficulties which are discussed in this section.

Compact Flash Card. At the beginning, several problems occurred with the CF card concerning data integrity. Some modifications (new distribution or files added) were not taken into account or caused problems during runtime. For instance, it was several times impossible to boot our distribution; the symbolic links did not once point to the good references; according to the sector used on the CF card, the display fluidity was not always the same. This problem was solved by formatting the CF card with the *fdisk* command before installing a new distribution and by invoking the command *lilo* each time a file was modified. Unfortunately lots of time was wasted because the causes of these events have not been easily determined at first.

Sound Disappearance. With each working distribution, the sound disappears after several tens of seconds. The sound reappears after pressing any scrolling key but this problem remains unresolved. It must be due again to the hardware limitations. Furthermore, the final system doesn't feature a keyboard, so the current hardware is undeniably not efficient enough.

Compilation Time and Integration Difficulties. The kernel and the mplayer compilations require a lot of time and is a source of discouragement when one spends time for compilation and realizes after that event that an option, a library or another thing has been missed. For instance, a lot of time has been spent to install framebuffer recompiling the kernel: it required about ten kernel recompilations so as to determine the right options and to disable those which are incompatible with the hardware. Unfortunately a lot of time has been wasted and as a consequence, some project sessions haven't been fruitful enough.

Framebuffer Mode. As just discussed before, a lot of time has been spent to make this mode run. As the Trident graphic card supports framebuffer, better performances could be expected and further software could be used by enabling this mode. Therefore, it was worth to spend time on it to get the best performances with the current limited hardware. However this mode only worked a week before the end of the project, so too few tests have been carried out.

Conclusion

The OM-Cube project keeps taking shape. Although we have come up against several difficulties due to the hardware, a first working distribution has been created. However, a more efficient hardware is needed to continue experimentations, with a TV output and an IR remote control. The current system enables to display several multimedia streams on a computer screen thanks to Mplayer and features a LCD screen but it remains impossible to navigate and choose the streams to play but DVDs. Framebuffer mode is used in order to get the most fluent display according to the hardware. Further work is needed, especially for IR communications and player skins which allow navigation and track selection.

This project has enabled us to learn lots of things related to the embedded world and Linux. So, even if we have not achieved the project by lack of time, it was a very profitable experience: it allowed us to handle source modification and compilation, to create a new Linux distribution, to drive a LCD screen and to adapt the existent tools to a specific hardware.

References

[1] OM-Cube bibliographical study, N. Marchand – V. Hiribarren – N. Talfer, 2003