The Linux Frame Buffer Device Subsystem

Geert Uytterhoeven
geert@linux-m68k.org
`http://www.cs.kuleuven.ac.be/~geert/`

ABSTRACT

The frame buffer device in recent kernels provides an abstraction for frame buffer based graphics hardware.

In this paper we will discuss two important aspects of frame buffer devices. Graphical consoles are implemented in the kernel on top of the frame buffer device. A second topic is the access of the graphics hardware by user applications. The most important application here is the (accelerated) X server.

1 Introduction

Many machines (m68k, SPARC, PowerMac) use graphical consoles because either the hardware does not support (VGA) text mode, or because the firmware programs the hardware into a graphical mode. Thus the Linux kernel needs to be aware of this and support graphical consoles on these architectures.

Graphical consoles will also gain more importance on the Intel (PC) platform, as VGA compatibility will be phased out by many graphics chipset manufacturers in the near future. An early example of this is the Cyrix MediaGX [1], which provides VGA compatibility through its BIOS only.

The frame buffer device provides an abstraction for the graphics hardware. It represents the frame buffer of some video hardware and allows application software to access the graphics hardware through a well-defined interface, so the software does not need to know about the low level (hardware register) stuff (except for hardware acceleration).

Since years there has been minor support for graphical consoles in the Linux kernel, but it differed a lot among the different platforms. Starting with kernel 2.1.107, the frame buffer device abstraction became completely integrated in the standard kernel sources and will become the standard way to access graphics hardware. But it is definitely not new: it originated from Linux/m68k at the end of 1994 and has proven to fulfill its task during the previous years.

There are two important aspects of frame buffer devices that will be discussed here:

Graphical consoles Emulation of a text console on a graphical display. This is kernel business.

User applications How to access the graphics hardware from user space?

2 Advantages of Frame Buffer Devices

The frame buffer device abstraction has the following advantages:

- It provides a unified method to access graphics hardware across different platforms.

- Drivers can be shared among different architectures, which reduces code duplication. There were already three different drivers for the ATI Mach64 before. In the ideal case, a frame buffer device driver contains a chipset driver core, with machine and bus dependent probe code (Zorro/PCI/ISA/Open Firmware/...).

- It provides simple multi-head: currently up to 8 frame buffer devices (displays) are supported. Unfortunately the input part of the console subsystem is not ready for multi-head yet.

- On boot up, you get one or more penguin logos (with or without beer :-) The more CPUs you have, the more penguins you will see.

3 The Frame Buffer Device Console

3.1 The Abstract Console Driver

The original Linux console driver was very intimately tied to VGA hardware and a VGA compatible text mode. To become more generic and powerful, and support all kinds of consoles, it had to be changed substantially. Herefore we designed the "abstract console driver". This is a small layer that provides an abstraction to a generic console.

A console needs to provide the following basic high level operations[1]:

putc Draw one character,

putcs Draw a string of characters,

scroll Scroll (a part of) the screen,

clear Clear a block of characters,

bmove Move a block of characters.

This scheme is much more flexible than the old `scr_{read,write}()` interface, which processed single characters. It can be mapped onto different kinds of consoles:

- Text consoles, e.g. VGA and MDA (vgacon and mdacon),

- Graphical consoles through the frame buffer device (fbcon),

- Graphics boards with a specialized graphical processor (e.g. TMS34010), where the host CPU does not have direct access to the frame buffer on the graphics board, only through a small window to the graphical processor (gspcon).

For graphical consoles, the high level console layer uses a shadow screen to remember which characters are shown where. On VGA text hardware, the overhead of a shadow screen is removed to keep the console fast (in fact it is faster than the old console driver in 2.0.x).

3.2 Design of the Frame Buffer Device Console

In the frame buffer device console driver, there is interaction between three parts (see Figure 1):

1. **fbdev** The frame buffer device that controls the hardware,

2. **fbcon** The generic frame buffer device console driver,

[1] Actually some more operations are needed, but they are not that important. Read `<linux/console.h>` for a complete list. [6]

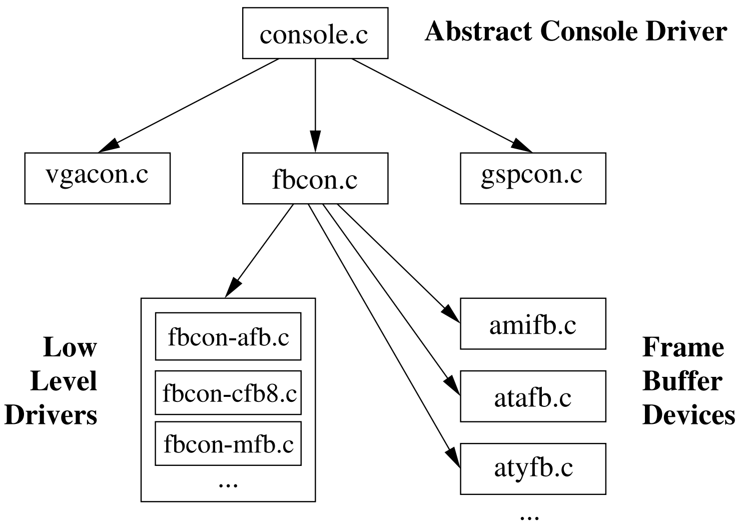

Figure 1: Interaction between the different parts of the console driver.

3. **fbcon-**$*$ Low level drawing routines for various screen organizations (chunky, planar, ...)[2].

The first and third components can be loadable modules, the second cannot (yet).

3.3 Performance of the Frame Buffer Device Console

Graphical consoles are inherently slower than hardware text consoles. However, they provide some features that cannot be provided by the classical VGA text console:

- Graphical consoles are more flexible. They can support arbitrary resolutions, color depths, refresh rates and font sizes. If the hardware supports the video mode you want, you can get it.

- Theoretically, there are no limits for the number of characters in a font.

- Graphical consoles can have fancy features like anti-aliased fonts.

To achieve the highest possible performance (especially for scrolling, which is slow when done using `memcpy()`) on various kinds of hardware, it is very important to use special features of the graphics hardware. Currently the frame buffer device console can make use of the following features, when available:

ywrap Scrolling can be sped up if the graphics hardware supports vertical wrap around of the screen (split screen). This is used on e.g. Amiga.

ypan If ywrap is not available, panning a large vertical virtual screen is another option. Only when the bottom (or top) of the virtual screen is reached, the screen contents have to be copied.

bitblt If the hardware supports accelerated rectcopy and rectfill operations, the graphics acceleration engine can be used for scrolling and clearing. This is done by replacing some low level drawing routines in

[2] See `struct display_switch` in `<video/fbcon.h>`. [6]

struct display_switch by accelerated routines.

smart redraw If reading from the frame buffer memory is slow, compared to writing, it makes sense to scroll the display by redrawing the characters that have changed only. Of course the scrolling speed then depends on the screen contents. This can even outperform the acceleration engine (which is used synchronuously) on machines with reasonable fast buses (e.g. PCI). For slow buses (e.g. ISA and Zorro II), the acceleration engine performs better.

real text mode If the graphics hardware has a hardware text mode, it may use it to speed up text consoles.

Benchmark results for ATI Mach64 GT (3D RAGE II+) are shown in Table 1. Benchmark data for smart redraw are not included, since the scrolling speed for smart redraw depends on the screen contents.

4 Using Frame Buffer Devices

4.1 Accessing Frame Buffer Devices

From user space, a frame buffer device can be accessed through a special device node (/dev/fb[0-7]), just like most other devices. These nodes provide the following operations:

normal I/O Reading and writing /dev/fb* gives access to the raw frame buffer contents.

mmap Using mmap() one can map the frame buffer into the memory space of the user program. After this the frame buffer becomes a piece of accessible memory.

Optionally one can map the MMIO (Memory Mapped I/O) registers from userspace. Note that you cannot mmap() these registers unless you have disabled hardware acceleration for the text console first, to protect against acceleration engine access conflicts.

ioctl Using ioctl() you can get or set the video mode, update the colormap, query the graphics chipset type, screen organization and visual type, ... Each frame buffer device can define additional ioctls for device specific operations, if necessary.

There is no generic hardware acceleration code in the kernel. All generic hardware acceleration operations have to be done from user space, using the mapped MMIO registers.

4.2 Applications using the Frame Buffer Device

- XF68_FBDev and XF86_FBDev, [5]
- ppmtofb: a simple picture viewer,
- the LibGGI fbdev target, [2]
- oFBis (part of OSIS [4]),
- fbset, a tool to manage video modes.

4.3 The (accelerated) X Server

The X server is the most important application for the frame buffer device. There exists an unaccelerated X server that uses the frame buffer device abstraction: XF68_FBDev[3].

[3]The "68" indicates the origin on Linux/m68k. Starting with XFree86 3.3.3, XF68_FBDev is used on big endian, and XF86_FBDev on little endian platforms

bpp	memcpy	bitblt	memcpy+ypan	bitblt+ypan
8	20.34s	1.66s	0.41s	0.19s
16	41.52s	3.75s	N/A	N/A
32	90.67s	10.99s	N/A	N/A

Table 1: Benchmark results for scrolling on a frame buffer console on the ATI 3D RAGE II+ (PCI, 4 MB SGRAM) in a CHRP/PPC box (604e at 200 MHz). The video mode was set to 1024x768, 75 Hz, using a 8x16 font. For ypan, the virtual screen size was 1024x4000.

It has been part of XFree86[5] since its integration in XFree86 release 3.2. The X server supports most popular screen organizations (monochrome, chunky 8/16/24/32 bits per pixel[4], and interleaved and non-interleaved bitplanes).

Work is in progress to add hardware acceleration to the X server. An experimental version for the NCR77C32BLT (used on the Amiga Retina Z3 graphics board), the ATI Mach64, the IMS TwinTurbo, the Matrox Millennium and the 3DLabs Permedia 2 is available.

For many chipsets, hardware acceleration can be based on the standard XFree86 acceleration code that already supports a zillion chipsets, and usually requires only minor modifications to the XFree86 code base. E.g. the accelerated X server for the Matrox Millennium differs from the normal XF86_SVGA server in the probe code only.

The main difference between an accelerated frame buffer device based X server and a "classical" X server is that video mode programming and colormap updates are now handled by the kernel. The X server still has to take care of acceleration engine setup and accelerated operations. Performance-wise there is no difference between the two approaches, if one considers the same hardware.

The upcoming XFree86 4.0 will have only one X server with loadable modules for different chipsets. Support for both systems with and without frame buffer devices is planned by having initialization code that uses the `ioctls` from the frame buffer API on systems that support them, and "hardware-banging" elsewhere.

[4]Due to endianness problems with the X11 cfb24 code, 24 bpp is available on little endian platforms only.

5 Portability Problems

5.1 Endianness: the Egg vs. the Egg

Endianness defines how objects that are larger than one byte are stored in memory. Traditionally there are two types of endianness: big endian (BE) and little endian (LE)[5]. E.g. the 32 bit number 0x12345678, which needs 4 bytes, is stored like:

	0	1	2	3
Big endian:	0x12	0x34	0x56	0x78

	0	1	2	3
Little endian:	0x78	0x56	0x34	0x12

The different components in a computer related to graphics processing (CPU, graphics display and acceleration engines, and system bus) can be either BE or LE. Graphics engines are usually configurable for both BE and LE systems. PCI, currently the most popular system bus, is LE by definition.

[5]To make things even more complicated, the PDP-11 is situated in between. But as long as there is no port of Linux to the PDP-11, we will not care about that.

If the endianness of the graphics chip and the CPU differ, this causes serious troubles, depending on the color depth:

8 **bpp** Each byte corresponds to one pixel, but the ordering of 4 consecutive pixels in a 32-bit word depends on the endianness.

$>$ 8 **bpp** One pixel spans multiple bytes.

$<$ 8 **bpp** Within each byte, there are multiple pixels, in an endianness-dependent order.

An example of what problems can show up is the CyberVision64 graphics boards on Amiga. While the m68k CPU in the Amiga is BE, the S3 Trio64 graphics chip on the CyberVision64 supports LE only. For 8 bpp, the problem was circumvented by physically swapping the data lines on the graphics board. However, there remains a problem for 16 bpp (e.g. 5 bits for R(ed), 6 for G(reen) and 5 for B(blue)):

	0	1
m68k	RRRRRGGG	GGGBBBBB
S3 Trio64	GGGBBBBB	RRRRRGGG

Thus the data has to be swapped when being transferred between CPU and frame buffer. Needless to say, 24 bpp becomes even more of a nightmare if the endianness does not match. The frame buffer console code can handle this, but X is a different beast.

5.2 MMIO Region Accesses

On some architectures, MMIO must be accessed differently from normal memory accesses. This becomes even more complex if you have multiple memory spaces on e.g. multiple PCI buses.

5.3 Advice for Graphics Chipset Designers

To avoid problems with support for your new state-of-the-art graphics chipset, I would like to ask to fulfill the following requirements:

- Please support both big and little endian machines. Intel is not the only player in the computer business.
- Please provide for a clean design with separate MMIO regions (at least on different pages, from the MMU point of view) for video mode programming (in the kernel) and acceleration (in user space) [7].
- Please support Open Source[3] software, and release programming information without nasty NDAs (Non-Disclosure Agreements).

... and your Linux customers will be happy :-)

6 Future Work

- More and better drivers. Currently more than 30 frame buffer devices have been written, but most of them are targeted at non-Intel platforms.
- Full acceleration for various chipsets using XFree86 code, and integration into the XFree86 branch that will result in XFree86 release 4.0.
- Real multi-head. This will require a significant rewrite of the console layer, to support multiple heads with multiple input devices.
- A better separation between frame buffer devices (fbdev) and the frame buffer device console driver (fbcon). On e.g. some embedded systems you may want to have

graphics support without the need for a text console.

some CPUs (e.g. m68k and PPC), all I/O is memory mapped.

Acknowledgments

- Martin Schaller designed the frame buffer device abstraction in 1994 and 1995. Unfortunately he disappeared from the scene completely.
- Martin Mares did a major rewrite of the high level console code after the abstract console code was integrated in kernel 2.1.107, and he made lots of optimizations.
- Jakub Jelinek made a lot of optimizations to the frame buffer console code.

And of course I would like to thank the numerous people who wrote and contributed code to frame buffer device drivers, and who gave suggestions and feedback.

Glossary

Frame buffer A block of memory that contains a representation of the screen image of a graphical display.

ISA Industry Standard Architecture. The legacy system bus that originated on the IBM PC XT (8 bit) and AT (16 bit), and runs at 8 MHz.

Open Firmware A FORTH-based firmware ("BIOS") used on some systems based on the SPARC and PPC CPUs.

MMIO Memory Mapped I/O. An I/O device can be accessed simply by reading from or writing to some addresses in memory, instead of using special I/O instructions. On

PCI Peripheral Component Interconnect. A system bus which exists in both 32 and 64 bit versions, and runs at 33 or 66 MHz.

VGA Video Graphics Array. A graphics hardware standard designed by IBM. It provides both hardware text modes and graphical modes. Most current video boards contain a VGA compatible core, but provide many extra features (higher resolutions and color depths).

Zorro The expansion bus used on Amiga. The Zorro bus comes in two flavors: Zorro II (16 bit, 8 MHz) and Zorro III (32 bit, 25 MHz).

References

[1] Cyrix MediaGX[tm] Processors. *http://www.cyrix.com/html/products/mediagx/index.htm*.

[2] General Graphics Interface. *http://www.ggi-project.org/*.

[3] Open Source. *http://www.opensource.org/*.

[4] OSIS: Atari TOS, Mint, AES, and VDI emulation for Linux/m68k. *http://www.nocrew.org/osis/*.

[5] The XFree86 Project, Inc. *http://www.xfree86.org/*.

[6] L. Torvalds. The Linux Kernel Sources. *ftp://ftp.kernel.org/pub/linux/v2.2/*.

[7] L. Vepstas. High Performance Graphics Hardware Design Requirements. *http://www.linas.org/linux/graphics.html*.