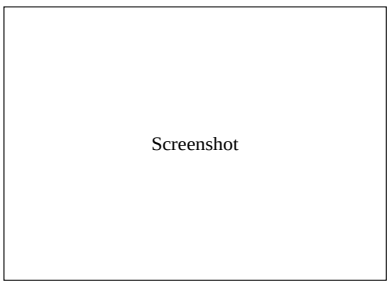
The Linux Frame Buffer Device Subsystem



Geert Uytterhoeven
geert@linux-m68k.org

Overview

◇ Introduction to Frame Buffer Devices

◇ The Abstract Console Driver

◇ The Frame Buffer Console Driver

◇ Frame Buffer Device Programming

◇ Applications using the Frame Buffer Device

◇ Conclusions and Future Development

Introduction

What? The frame buffer device

◇ provides an abstraction for the graphics hardware

◇ represents the frame buffer of the graphics hardware

◇ provides a well-defined interface to the hardware for application software

Exception: hardware acceleration is hardware specific

Why?

→ Need for **graphical text consoles**

Why Graphical Text Consoles?

- Non-PC hardware doesn't always support (VGA) text mode
- Firmware may program the hardware in graphics mode
- VGA legacy support will be removed eventually

Other features:

+ More flexible: arbitrary resolutions, color depths, refresh rates, font sizes
+ No limit for the number of characters in a font
+ Fancy features like anti-aliased fonts, . . .
- Inherently slower

History of Frame Buffer Devices

- Linux was developed for PC hardware
  MDA → CGA → EGA → VGA: hardware text mode
- 1993: Linux/m68k (initially Amiga & Atari)
  No hardware textmode, bitmapped graphics only
  Abstraction: `drivers/char/console.c` → `amicon.c` and `atacon.c`
- 1994: X through the **frame buffer device** (Martin Schaller on Atari)
  - **map frame buffer: `mmap()`**
  - **query resolution: `ioctl()`**
  - **change colormap: `ioctl()`**
- Merge `amicon.c` and `atacon.c` into `fbcon.c`
  Other m68k ports emerged: Mac, Apollo, . . . → more frame buffer devices

History of Frame Buffer Devices (cont'd)

- Other architectures? AXP, SPARC, PPC (PowerMac)
  - Hook into `scr_readw()`, `scr_writew()`
  - Shadow screen
  - Smart redraw
- Merge m68k with main branch (Linus)
  - Abstract console driver: fbcon and vgacon
  - Port to PPC (offb, atyfb), AXP (tgafb) and ia32 (vesafb)
  - Kernel 2.1.107: **frame buffer console = standard method for graphical consoles under Linux**
- Future: removal of VGA legacy (e.g. Cyrix MediaGX) → important for ia32, too

Design Goals for the Abstract Console Driver

Flexibility: support all kinds of consoles, no performance loss

- Hardware text consoles (VGA, MDA)
- Graphical consoles through frame buffer devices
- Graphical Signal Processors (GSP) without direct frame buffer access, e.g. TMS34010
- . . .

Implementation of the Abstract Console Driver

Small layer that provides an abstraction to a generic console.

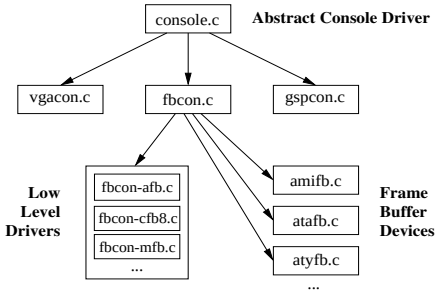
Primitives (`include/linux/console.h`):

```
struct consw {
    void (*con_clear)(...);
    void (*con_putc)(...);
    void (*con_putcs)(...);
    int  (*con_scroll)(...);
    void (*con_bmove)(...);
};
```

- Shadowscreen (optimization: not for VGA/MDA)
- Scrollback

The Frame Buffer Device Console

1. **fbdev**: frame buffer device controls the hardware
2. **fbcon**: generic frame buffer device console driver
3. **fbcon-**$*$: low level drawing routines for various screen memory organizations



Text Console Acceleration

- Use hardware tricks for acceleration
  - ypan: large virtual screen
  - ywrap: wrap around (← `amicon.c`, using the Amiga copper)
  - 2D acceleration engine (replace generic drawing operations)
  - real hardware text mode
- Smart redraw
  - Draw changed characters only
  - Reading the frame buffer is usually slow
  - PCI is fast, busy wait for blitter

Text Scrolling Benchmark Results

- ATI Mach64 GT (3D RAGE II+), PCI, 4 MB SGRAM
- CHRP LongTrail with IBM 604E at 200 MHz
- 1024×768, 75 Hz, 8×16 font
- ypan: virtual screen 1024×4000

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

Note: no results for smart redraw

Console Initialization

- We want console output as early as possible
- VGA text console is available early (on ISA bus, behind PCI/ISA bridge)
- Frame buffer devices need bus probing and memory allocation
- Solution: `dummy_con` and `take_over_console()`

```
conswitchp = &dummy_con;
console_init();
mem_init();
<bus>_init();
take_over_console();
```

- Allows booting with vgacon, and loading frame buffer device modules later

The Frame Buffer Device API

Special character device nodes: `/dev/fb[0-7]`

- **File I/O**: Frame buffer contents
- **`ioctl()`**: Query/change properties
- **`mmap()`**: Access to the **frame buffer** and the **MMIO registers**

```
include/linux/fb.h
include/video/
Documentation/fb/
http://www.tahallah.demon.co.uk/programming/prog.html
```


The Frame Buffer Device API: `ioctl()`

`FBIOGET_FSCREENINFO`

Query unchangeable information about the hardware

```
struct fb_fix_screeninfo {
    char id[16];
    char *smem_start;
    __u32 smem_len;
    __u32 type;
    __u32 visual;
    char *mmio_start;
    __u32 mmio_len;
    __u32 accel;
};
```


The Frame Buffer Device API: `ioctl()`

`FBIOGET_VSCREENINFO`, **`FBIOPUT_VSCREENINFO`**

Query or change variable information about the hardware

```
struct fb_var_screeninfo {
    __u32 xres, yres, xres_virtual, yres_virtual;
    __u32 xoffset, yoffset;
    __u32 bits_per_pixel;
    struct fb_bitfield red, green, blue, transp;
    __u32 pixclock;
    __u32 left_margin, right_margin, upper_margin,
          lower_margin;
    __u32 hsync_len, vsync_len;
};
```

Driver may round up some values to meet the hardware's capabilities

The Frame Buffer Device API: `ioctl()`

`FBIOGETCMAP`, **`FBIOPUTCMAP`**

Request or change parts of the color map

```
struct fb_cmap {
    __u32 start, len;
    __u16 *red, *green, *blue, *transp;
};
```

+ frame buffer device specific `ioctl()`s

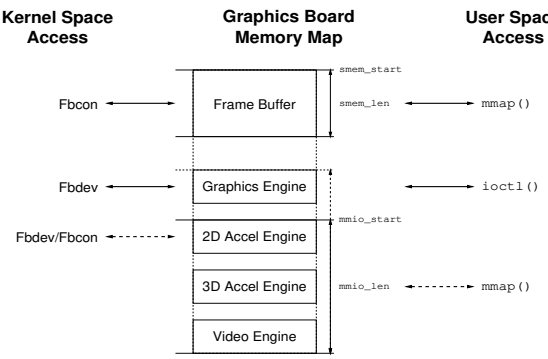
A Sample Application

```
fd = open("/dev/fb0", O_RDWR);
ioctl(fd, FBIOGET_VSCREENINFO, &var);
ioctl(fd, FBIOGET_FSCREENINFO, &fix);
cmap.start = 0;
cmap.len = 256;
cmap.red = malloc(cmap.len*sizeof(*cmap.red));
cmap.green = malloc(cmap.len*sizeof(*cmap.green));
cmap.blue = malloc(cmap.len*sizeof(*cmap.blue));
cmap.transp = malloc(cmap.len*sizeof(*cmap.transp));
ioctl(fd, FBIOGETCMAP, cmap);
line = malloc(fix.line_length);
printf("P6\n%d %d\n255\n", var.xres, var.yres);
for (i = 0; i < var.yres; i++) {
    read(fd, line, fix.line_length);
    for (j = 0; j < var.xres; j++) {
        putc(cmap.red[(int)line[j]]>>8, stdout);
        putc(cmap.green[(int)line[j]]>>8, stdout);
        putc(cmap.blue[(int)line[j]]>>8, stdout);
    }
}
close(fd);
```


The Frame Buffer Device API: `mmap()`

```
fd = open("/dev/fb0", O_RDWR);
ioctl(fd, FBIOGET_FSCREENINFO, &fix);
fb = mmap(NULL, fix.smem_start, fix.smem_len,
          PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
ioctl(fd, FBIOGET_VSCREENINFO, &var);
var.accel &= ~FB_ACCELF_TEXT;
ioctl(fd, FBIOPUT_VSCREENINFO, &var);
mmio = mmap(NULL, fix.mmio_start, fix.mmio_len,
            PROT_READ | PROT_WRITE, MAP_SHARED,
            fd, fix.smem_len);
...
munmap(mmio, fix.mmio_len);
munmap(fb, fix.smem_len);
close(fd);
```


Graphics Hardware Access



Hardware Acceleration

No generic hardware acceleration support in the kernel

- Text console acceleration (rectcopy and rectfill) is allowed
- Hardware acceleration from user space, using `mmap()` on the Memory Mapped I/O (MMIO) registers

Motivations:

- User space acceleration experience (e.g. XFree86)
- Hardware acceleration is hardware specific
- Generic hardware acceleration is be complex → no OpenGL in the kernel

Future: 3D pipelines under kernel control? User space DMA?

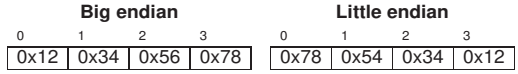
Multi-Head

- Up to 8 frame buffer devices
- Map any VC to any fbdev: **`ioctl()`s**

  `FBIOGET_CON2FBMAP`
  `FBIOPUT_CON2FBMAP`
- Multiple input devices? (PS/2, ADB, USB, Access.Bus, . . .)
- Rewrite of the input layer needed

Endianness

0x12345678



- CPU: big or little endian
- Graphics engine: usually configurable
- System bus: PCI is little endian

Example: 16 bpp little endian in a big endian machine (CyberVision64 on Amiga)

Traditional Graphics Hardware Access under Linux

PC hardware (VGA compatible)

- Kernel (virtual text consoles): VGA compatible hardware text mode (+ SVGATextMode)
- User space graphics:
  - SVGAlib (limited support)
  - X server (XFree86, Xi Graphics, Metro Link)

Advantages of the Frame Buffer Device Abstraction

◇ Unified method to access graphics hardware across different platforms

◇ Drivers can be shared among different architectures → reduces code duplication

Frame buffer device = chipset driver core + machine dependent probe code

◇ Simple multi-head: up to 8 frame buffer devices (displays)

◇ (with or without beer :-), one for each CPU

Special Case: the IA32 Platform

- vesafb: requires VESA BIOS extensions 2.0
- vga16fb: all VGA compatible hardware
- Chipset specific frame buffer devices, currently
  - atyfb: ATI Mach64
  - matroxfb: Matrox Millennium
  - pm2fb: 3DLabs Permedia 2
  - sgivwfb: SGI Visual Workstation

Quick Frame Buffer Device Writer's HOWTO

Need to know first :

- How to program a video mode (unless done by Firmware, cfr. `offb`)
- Where's the frame buffer?
- How to change the colormap?

Good starting point: `drivers/video/tgafb.c`
→ X works immediately (XF{68,86}_FBDev)

Later :

- Dynamic video mode changing
- Rectfill and rectcopy for text console acceleration
- Hardware cursor

Applications using the Frame Buffer Device

- XF68_FBDev and XF86_FBDev: X servers from the XFree86 Project
- fbset: tool to manage video modes, database in `/etc/fb.modes`
- ppmtofb: simple picture viewer
- libGGI `fbdev` target from the General Graphics Interface (GGI) Project
- oFBis (part of OSIS)

The X server

→ **XFree86** (commercial X servers?)

- XF68_FBDev: integrated since XFree86 release 3.2
  - monochrome
  - chunky 8/16/24/32 bpp
  - non-interleaved bitplanes
  - interleaved bitplanes
- Hardware acceleration? Experimental patches for XFree86 release 3.3.3.1
  - NCR77C32BLT
  - IMS Twin Turbo
  - 3DLabs Permedia 2
  - ATI Mach64
  - Matrox Millennium
- Better: integrate in existing accelerated X servers (cfr. Matrox in XF86_SVGA)

The X server

	Traditional	Fbdev
Hardware probing	Direct access	`/dev/fb* ioctl()` `FBIOGET_FSCREENINFO`
Video mode switching	Direct access	`/dev/fb* ioctl()` `FBIOPUT_VSCREENINFO`
Colormap changing	Direct access	`/dev/fb* ioctl()` `FBIOPUTCMAP`
Acceleration	Direct access	Direct access

→ **XFree86 release 4.0**: one single X server

- Use frame buffer device probing if available
- Else fallback to direct hardware probing
- Common acceleration code, no performance loss

The GGI Project

GGI = **General Graphics Interface**

- libGGI `fbdev` target
- GGI drivers masquerade as frame buffer devices

Future Development

- More and better drivers
  Currently >30 frame buffer devices, most targeted at non-ia32 platforms
- Full acceleration for various chipsets using XFree86 code
  → XFree86 release 4.0
- Real multi-head → rewrite of the input layer
- Better separation between fbdev and fbcon
- Shared video mode database to select the video mode at boot up time

Development mailing list (under majordomo):

`linux-fbdev@vuser.vu.union.edu`

Conclusions

fbdev

- Frame buffer devices provide a good abstraction for the graphics hardware
- Accelerated X servers on top of frame buffer devices are possible and perform well

abscon

- The abstract console driver allows all kinds of consoles, without performance loss

fbcon

- The frame buffer console driver provides a fast text console on graphics hardware

Acknowledgements

Jose Alonso	Martin Apel	Andrew Apted	Alan Bair
Philip Blundell	Thomas Bogendörfer	Alex Buell	David Carter
Martin Costabel	Alan Cox	Peter De Schrijver	Eddie C. Dost
Michel Dänzer	Arno Griffioen	Bernd Harries	Benjamin Herrenschmidt
Roman Hodek	Dan Jacobowitz	Jakub Jelinek	Günther Kelleter
David D. Kilzer	Russell King	Gerd Knorr	Mike Lockwood
Alexander V. Lukyanov	Hamish Macdonald	Pavel Machek	Paul Mackerras
Martin Mares	David S. Miller	Ilario Nardinocchi	Barry Nathan
Frank Neumann	Jeffrey Newquist	Ryan Nielsen	Ryuichi Oikawa
Jürgen Orschiedt	Ben Pfaff	Gleb Raiko	Peter Remmers
Tom Rini	Vladimir Roganov	Tim Rowley	William Rucklidge
Jeff Rugen	Antonio Santos	Martin Schaller	Bernd Schmidt
Michael Schmitz	Kevin Schoedel	Andreas Schwab	Franz Sirl
Jes Sørensen	Michael Teske	Randy Thelen	Geert Uytterhoeven
Petr Vandrovec	Brian K. White	Pete Zaitcev	Richard Zidlicky
Roman Zippel	...		

My Future?

Linux kernel hacker for hire!

geert@linux-m68k.org

http://www.cs.kuleuven.ac.be/~geert/