Linux Fb Documentation

Release 6.8.0

The kernel development community

Jan 16, 2026

CONTENTS

CHAPTER

ONE

THE FRAME BUFFER DEVICE API

Last revised: June 21, 2011

1.1 0. Introduction

This document describes the frame buffer API used by applications to interact with frame buffer devices. In-kernel APIs between device drivers and the frame buffer core are not described.

Due to a lack of documentation in the original frame buffer API, drivers behaviours differ in subtle (and not so subtle) ways. This document describes the recommended API implementation, but applications should be prepared to deal with different behaviours.

1.2 1. Capabilities

Device and driver capabilities are reported in the fixed screen information capabilities field:

```
struct fb_fix_screeninfo {
      ...
      __u16 capabilities;             /* see FB_CAP_*                 */
      ...
};
```

Application should use those capabilities to find out what features they can expect from the device and driver.

- FB_CAP_FOURCC

The driver supports the four character code (FOURCC) based format setting API. When supported, formats are configured using a FOURCC instead of manually specifying color components layout.

1.3 2. Types and visuals

Pixels are stored in memory in hardware-dependent formats. Applications need to be aware of the pixel storage format in order to write image data to the frame buffer memory in the format expected by the hardware.

Formats are described by frame buffer types and visuals. Some visuals require additional information, which are stored in the variable screen information bits_per_pixel, grayscale, red, green, blue and transp fields.

Visuals describe how color information is encoded and assembled to create macropixels. Types describe how macropixels are stored in memory. The following types and visuals are supported.

- FB_TYPE_PACKED_PIXELS

Macropixels are stored contiguously in a single plane. If the number of bits per macropixel is not a multiple of 8, whether macropixels are padded to the next multiple of 8 bits or packed together into bytes depends on the visual.

Padding at end of lines may be present and is then reported through the fixed screen information line_length field.

- FB_TYPE_PLANES

Macropixels are split across multiple planes. The number of planes is equal to the number of bits per macropixel, with plane i'th storing i'th bit from all macropixels.

Planes are located contiguously in memory.

- FB_TYPE_INTERLEAVED_PLANES

Macropixels are split across multiple planes. The number of planes is equal to the number of bits per macropixel, with plane i'th storing i'th bit from all macropixels.

Planes are interleaved in memory. The interleave factor, defined as the distance in bytes between the beginning of two consecutive interleaved blocks belonging to different planes, is stored in the fixed screen information type_aux field.

- FB_TYPE_FOURCC

Macropixels are stored in memory as described by the format FOURCC identifier stored in the variable screen information grayscale field.

- FB_VISUAL_MONO01

Pixels are black or white and stored on a number of bits (typically one) specified by the variable screen information bpp field.

Black pixels are represented by all bits set to 1 and white pixels by all bits set to 0. When the number of bits per pixel is smaller than 8, several pixels are packed together in a byte.

FB_VISUAL_MONO01 is currently used with FB_TYPE_PACKED_PIXELS only.

- FB_VISUAL_MONO10

Pixels are black or white and stored on a number of bits (typically one) specified by the variable screen information bpp field.

Black pixels are represented by all bits set to 0 and white pixels by all bits set to 1. When the number of bits per pixel is smaller than 8, several pixels are packed together in a byte.

FB_VISUAL_MONO01 is currently used with FB_TYPE_PACKED_PIXELS only.

- FB_VISUAL_TRUECOLOR

Pixels are broken into red, green and blue components, and each component indexes a read-only lookup table for the corresponding value. Lookup tables are device-dependent, and provide linear or non-linear ramps.

Each component is stored in a macropixel according to the variable screen information red, green, blue and transp fields.

- FB_VISUAL_PSEUDOCOLOR and FB_VISUAL_STATIC_PSEUDOCOLOR

Pixel values are encoded as indices into a colormap that stores red, green and blue components. The colormap is read-only for FB_VISUAL_STATIC_PSEUDOCOLOR and read-write for FB_VISUAL_PSEUDOCOLOR.

Each pixel value is stored in the number of bits reported by the variable screen information bits_per_pixel field.

- FB_VISUAL_DIRECTCOLOR

Pixels are broken into red, green and blue components, and each component indexes a programmable lookup table for the corresponding value.

Each component is stored in a macropixel according to the variable screen information red, green, blue and transp fields.

- FB_VISUAL_FOURCC

Pixels are encoded and interpreted as described by the format FOURCC identifier stored in the variable screen information grayscale field.

1.4 3. Screen information

Screen information are queried by applications using the FBIOGET_FSCREENINFO and FBIOGET_VSCREENINFO ioctls. Those ioctls take a pointer to a fb_fix_screeninfo and fb_var_screeninfo structure respectively.

struct fb_fix_screeninfo stores device independent unchangeable information about the frame buffer device and the current format. Those information can't be directly modified by applications, but can be changed by the driver when an application modifies the format:

```
struct fb_fix_screeninfo {
      char id[16];                    /* identification string eg "TT Builtin"␣
↪*/
      unsigned long smem_start;       /* Start of frame buffer mem */
                                      /* (physical address) */
      __u32 smem_len;                 /* Length of frame buffer mem */
      __u32 type;                     /* see FB_TYPE_*                */
      __u32 type_aux;                 /* Interleave for interleaved Planes */
      __u32 visual;                   /* see FB_VISUAL_*              */
      __u16 xpanstep;                 /* zero if no hardware panning  */
      __u16 ypanstep;                 /* zero if no hardware panning  */
      __u16 ywrapstep;                /* zero if no hardware ywrap    */
      __u32 line_length;              /* length of a line in bytes    */
```

```
      unsigned long mmio_start;       /* Start of Memory Mapped I/O   */
                                      /* (physical address) */
      __u32 mmio_len;                 /* Length of Memory Mapped I/O  */
      __u32 accel;                    /* Indicate to driver which     */
                                      /*  specific chip/card we have  */
      __u16 capabilities;             /* see FB_CAP_*                 */
      __u16 reserved[2];              /* Reserved for future compatibility */
};
```

struct fb_var_screeninfo stores device independent changeable information about a frame buffer device, its current format and video mode, as well as other miscellaneous parameters:

```
struct fb_var_screeninfo {
      __u32 xres;                     /* visible resolution           */
      __u32 yres;
      __u32 xres_virtual;             /* virtual resolution           */
      __u32 yres_virtual;
      __u32 xoffset;                  /* offset from virtual to visible */
      __u32 yoffset;                  /* resolution                   */

      __u32 bits_per_pixel;           /* guess what                   */
      __u32 grayscale;                /* 0 = color, 1 = grayscale,    */
                                      /* >1 = FOURCC                  */
      struct fb_bitfield red;         /* bitfield in fb mem if true color, */
      struct fb_bitfield green;       /* else only length is significant */
      struct fb_bitfield blue;
      struct fb_bitfield transp;      /* transparency                 */

      __u32 nonstd;                   /* != 0 Non standard pixel format */

      __u32 activate;                 /* see FB_ACTIVATE_*            */

      __u32 height;                   /* height of picture in mm    */
      __u32 width;                    /* width of picture in mm     */

      __u32 accel_flags;              /* (OBSOLETE) see fb_info.flags */

      /* Timing: All values in pixclocks, except pixclock (of course) */
      __u32 pixclock;                 /* pixel clock in ps (pico seconds) */
      __u32 left_margin;              /* time from sync to picture    */
      __u32 right_margin;             /* time from picture to sync    */
      __u32 upper_margin;             /* time from sync to picture    */
      __u32 lower_margin;
      __u32 hsync_len;                /* length of horizontal sync    */
      __u32 vsync_len;                /* length of vertical sync      */
      __u32 sync;                     /* see FB_SYNC_*                */
      __u32 vmode;                    /* see FB_VMODE_*               */
      __u32 rotate;                   /* angle we rotate counter clockwise */
      __u32 colorspace;               /* colorspace for FOURCC-based modes */
      __u32 reserved[4];              /* Reserved for future compatibility */
};
```

To modify variable information, applications call the FBIOPUT_VSCREENINFO ioctl with a pointer to a fb_var_screeninfo structure. If the call is successful, the driver will update the fixed screen information accordingly.

Instead of filling the complete fb_var_screeninfo structure manually, applications should call the FBIOGET_VSCREENINFO ioctl and modify only the fields they care about.

1.5 4. Format configuration

Frame buffer devices offer two ways to configure the frame buffer format: the legacy API and the FOURCC-based API.

The legacy API has been the only frame buffer format configuration API for a long time and is thus widely used by application. It is the recommended API for applications when using RGB and grayscale formats, as well as legacy non-standard formats.

To select a format, applications set the fb_var_screeninfo bits_per_pixel field to the desired frame buffer depth. Values up to 8 will usually map to monochrome, grayscale or pseudocolor visuals, although this is not required.

- For grayscale formats, applications set the grayscale field to one. The red, blue, green and transp fields must be set to 0 by applications and ignored by drivers. Drivers must fill the red, blue and green offsets to 0 and lengths to the bits_per_pixel value.
- For pseudocolor formats, applications set the grayscale field to zero. The red, blue, green and transp fields must be set to 0 by applications and ignored by drivers. Drivers must fill the red, blue and green offsets to 0 and lengths to the bits_per_pixel value.
- For truecolor and directcolor formats, applications set the grayscale field to zero, and the red, blue, green and transp fields to describe the layout of color components in memory:

```
struct fb_bitfield {
    __u32 offset;                   /* beginning of bitfield        */
    __u32 length;                   /* length of bitfield           */
    __u32 msb_right;                /* != 0 : Most significant bit is */
                                    /* right */
};
```

  Pixel values are bits_per_pixel wide and are split in non-overlapping red, green, blue and alpha (transparency) components. Location and size of each component in the pixel value are described by the fb_bitfield offset and length fields. Offset are computed from the right.

  Pixels are always stored in an integer number of bytes. If the number of bits per pixel is not a multiple of 8, pixel values are padded to the next multiple of 8 bits.

Upon successful format configuration, drivers update the fb_fix_screeninfo type, visual and line_length fields depending on the selected format.

The FOURCC-based API replaces format descriptions by four character codes (FOURCC). FOURCCs are abstract identifiers that uniquely define a format without explicitly describing it. This is the only API that supports YUV formats. Drivers are also encouraged to implement the FOURCC-based API for RGB and grayscale formats.

Drivers that support the FOURCC-based API report this capability by setting the FB_CAP_FOURCC bit in the fb_fix_screeninfo capabilities field.

FOURCC definitions are located in the linux/videodev2.h header. However, and despite starting with the V4L2_PIX_FMT_prefix, they are not restricted to V4L2 and don't require usage of the V4L2 subsystem. FOURCC documentation is available in Documentation/userspace-api/media/v4l/pixfmt.rst.

To select a format, applications set the grayscale field to the desired FOURCC. For YUV formats, they should also select the appropriate colorspace by setting the colorspace field to one of the colorspaces listed in linux/videodev2.h and documented in Documentation/userspace-api/media/v4l/colorspaces.rst.

The red, green, blue and transp fields are not used with the FOURCC-based API. For forward compatibility reasons applications must zero those fields, and drivers must ignore them. Values other than 0 may get a meaning in future extensions.

Upon successful format configuration, drivers update the fb_fix_screeninfo type, visual and line_length fields depending on the selected format. The type and visual fields are set to FB_TYPE_FOURCC and FB_VISUAL_FOURCC respectively.

CHAPTER

TWO

ARKFB - FBDEV DRIVER FOR ARK LOGIC CHIPS

2.1 Supported Hardware

ARK 2000PV chip ICS 5342 ramdac

- only BIOS initialized VGA devices supported
- probably not working on big endian

2.2 Supported Features

- 4 bpp pseudocolor modes (with 18bit palette, two variants)
- 8 bpp pseudocolor mode (with 18bit palette)
- 16 bpp truecolor modes (RGB 555 and RGB 565)
- 24 bpp truecolor mode (RGB 888)
- 32 bpp truecolor mode (RGB 888)
- text mode (activated by bpp = 0)
- doublescan mode variant (not available in text mode)
- panning in both directions
- suspend/resume support

Text mode is supported even in higher resolutions, but there is limitation to lower pixclocks (i got maximum about 70 MHz, it is dependent on specific hardware). This limitation is not enforced by driver. Text mode supports 8bit wide fonts only (hardware limitation) and 16bit tall fonts (driver limitation). Unfortunately character attributes (like color) in text mode are broken for unknown reason, so its usefulness is limited.

There are two 4 bpp modes. First mode (selected if nonstd == 0) is mode with packed pixels, high nibble first. Second mode (selected if nonstd == 1) is mode with interleaved planes (1 byte interleave), MSB first. Both modes support 8bit wide fonts only (driver limitation).

Suspend/resume works on systems that initialize video card during resume and if device is active (for example used by fbcon).

2.3 Missing Features

(alias TODO list)

- secondary (not initialized by BIOS) device support
- big endian support
- DPMS support
- MMIO support
- interlaced mode variant
- support for fontwidths != 8 in 4 bpp modes
- support for fontheight != 16 in text mode
- hardware cursor
- vsync synchronization
- feature connector support
- acceleration support (8514-like 2D)

2.4 Known bugs

- character attributes (and cursor) in text mode are broken

-- Ondrej Zajicek <santiago@crfreenet.org>

CHAPTER

THREE

WHAT IS ATY128FB?

This is a driver for a graphic framebuffer for ATI Rage128 based devices on Intel and PPC boxes.

Advantages:

- It provides a nice large console (128 cols + 48 lines with 1024x768) without using tiny, unreadable fonts.
- You can run XF68_FBDev on top of /dev/fb0
- Most important: boot logo :-)

Disadvantages:

- graphic mode is slower than text mode... but you should not notice if you use same resolution as you used in textmode.
- still experimental.

3.1 How to use it?

Switching modes is done using the video=aty128fb:<resolution>... modedb boot parameter or using *fbset* program.

See *modedb default video mode support* for more information on modedb resolutions.

You should compile in both vgacon (to boot if you remove your Rage128 from box) and aty128fb (for graphics mode). You should not compile-in vesafb unless you have primary display on non-Rage128 VBE2.0 device (see *What is vesafb?* for details).

3.2 X11

XF68_FBDev should generally work fine, but it is non-accelerated. As of this document, 8 and 32bpp works fine. There have been palette issues when switching from X to console and back to X. You will have to restart X to fix this.

3.3 Configuration

You can pass kernel command line options to vesafb with *video=aty128fb:option1,option2:value2,option3* (multiple options should be separated by comma, values are separated from options by *:*). Accepted options:

noaccel	do not use acceleration engine. It is default.
accel	use acceleration engine. Not finished.
vmode:x	chooses PowerMacintosh video mode <x>. Deprecated.
cmode:x	chooses PowerMacintosh colour mode <x>. Deprecated.
<XxX@X>	selects startup videomode. See *modedb default video mode support* for detailed explanation. Default is 640x480x8bpp.

3.4 Limitations

There are known and unknown bugs, features and misfeatures. Currently there are following known bugs:

- This driver is still experimental and is not finished. Too many bugs/errata to list here.

Brad Douglas <brad@neruo.com>

CHAPTER

FOUR

FRAMEBUFFER DRIVER FOR CIRRUS LOGIC CHIPSETS

Chip families supported:

- SD64
- Piccolo
- Picasso
- Spectrum
- Alpine (GD-543x/4x)
- Picasso4 (GD-5446)
- GD-5480
- Laguna (GD-546x)

Bus's supported:

- PCI
- Zorro

Architectures supported:

- i386
- Alpha
- PPC (Motorola Powerstack)
- m68k (Amiga)

4.1 Default video modes

At the moment, there are two kernel command line arguments supported:

- mode:640x480
- mode:800x600
- mode:1024x768

Full support for startup video modes (modedb) will be integrated soon.

4.2 Version 1.9.9.1

- Fix memory detection for 512kB case
- 800x600 mode
- Fixed timings
- Hint for AXP: Use -accel false -vyres -1 when changing resolution

4.3 Version 1.9.4.4

- Preliminary Laguna support
- Overhaul color register routines.
- Associated with the above, console colors are now obtained from a LUT called 'palette' instead of from the VGA registers. This code was modelled after that in atyfb and matroxfb.
- Code cleanup, add comments.
- Overhaul SR07 handling.
- Bug fixes.

4.4 Version 1.9.4.3

- Correctly set default startup video mode.
- Do not override ram size setting. Define CLGEN_USE_HARDCODED_RAM_SETTINGS if you _do_ want to override the RAM setting.
- Compile fixes related to new 2.3.x IORESOURCE_IO[PORT] symbol changes.
- Use new 2.3.x resource allocation.
- Some code cleanup.

4.5 Version 1.9.4.2

- Casting fixes.
- Assertions no longer cause an oops on purpose.
- Bug fixes.

4.6 Version 1.9.4.1

- Add compatibility support. Now requires a 2.1.x, 2.2.x or 2.3.x kernel.

4.7 Version 1.9.4

- Several enhancements, smaller memory footprint, a few bugfixes.
- Requires kernel 2.3.14-pre1 or later.

4.8 Version 1.9.3

- Bundled with kernel 2.3.14-pre1 or later.

CHAPTER

FIVE

UNDERSTANDING FBDEV'S CMAP

These notes explain how X's dix layer uses fbdev's cmap structures.

- example of relevant structures in fbdev as used for a 3-bit grayscale cmap:

```
struct fb_var_screeninfo {
        .bits_per_pixel = 8,
        .grayscale      = 1,
        .red =          { 4, 3, 0 },
        .green =        { 0, 0, 0 },
        .blue =         { 0, 0, 0 },
}
struct fb_fix_screeninfo {
        .visual =       FB_VISUAL_STATIC_PSEUDOCOLOR,
}
for (i = 0; i < 8; i++)
    info->cmap.red[i] = (((2*i)+1)*(0xFFFF))/16;
memcpy(info->cmap.green, info->cmap.red, sizeof(u16)*8);
memcpy(info->cmap.blue, info->cmap.red, sizeof(u16)*8);
```

- X11 apps do something like the following when trying to use grayscale:

```
for (i=0; i < 8; i++) {
    char colorspec[64];
    memset(colorspec,0,64);
    sprintf(colorspec, "rgb:%x/%x/%x", i*36,i*36,i*36);
    if (!XParseColor(outputDisplay, testColormap, colorspec, &wantedColor))
            printf("Can't get color %s\n",colorspec);
    XAllocColor(outputDisplay, testColormap, &wantedColor);
    grays[i] = wantedColor;
}
```

There's also named equivalents like gray1..x provided you have an rgb.txt.

Somewhere in X's callchain, this results in a call to X code that handles the colormap. For example, Xfbdev hits the following:

xc-011010/programs/Xserver/dix/colormap.c:

```
FindBestPixel(pentFirst, size, prgb, channel)

dr = (long) pent->co.local.red - prgb->red;
```

```
dg = (long) pent->co.local.green - prgb->green;
db = (long) pent->co.local.blue - prgb->blue;
sq = dr * dr;
UnsignedToBigNum (sq, &sum);
BigNumAdd (&sum, &temp, &sum);
```

co.local.red are entries that were brought in through FBIOGETCMAP which come directly from the info->cmap.red that was listed above. The prgb is the rgb that the app wants to match to. The above code is doing what looks like a least squares matching function. That's why the cmap entries can't be set to the left hand side boundaries of a color range.

CHAPTER

SIX

DEFERRED IO

Deferred IO is a way to delay and repurpose IO. It uses host memory as a buffer and the MMU pagefault as a pretrigger for when to perform the device IO. The following example may be a useful explanation of how one such setup works:

- userspace app like Xfbdev mmaps framebuffer
- deferred IO and driver sets up fault and page_mkwrite handlers
- userspace app tries to write to mmapped vaddress
- we get pagefault and reach fault handler
- fault handler finds and returns physical page
- we get page_mkwrite where we add this page to a list
- schedule a workqueue task to be run after a delay
- app continues writing to that page with no additional cost. this is the key benefit.
- the workqueue task comes in and mkcleans the pages on the list, then completes the work associated with updating the framebuffer. this is the real work talking to the device.
- app tries to write to the address (that has now been mkcleaned)
- get pagefault and the above sequence occurs again

As can be seen from above, one benefit is roughly to allow bursty framebuffer writes to occur at minimum cost. Then after some time when hopefully things have gone quiet, we go and really update the framebuffer which would be a relatively more expensive operation.

For some types of nonvolatile high latency displays, the desired image is the final image rather than the intermediate stages which is why it's okay to not update for each write that is occurring.

It may be the case that this is useful in other scenarios as well. Paul Mundt has mentioned a case where it is beneficial to use the page count to decide whether to coalesce and issue SG DMA or to do memory bursts.

Another one may be if one has a device framebuffer that is in an usual format, say diagonally shifting RGB, this may then be a mechanism for you to allow apps to pretend to have a normal framebuffer but reswizzle for the device framebuffer at vsync time based on the touched pagelist.

6.1 How to use it: (for applications)

No changes needed. mmap the framebuffer like normal and just use it.

6.2 How to use it: (for fbdev drivers)

The following example may be helpful.

1. Setup your structure. Eg:

```
static struct fb_deferred_io hecubafb_defio = {
        .delay          = HZ,
        .deferred_io    = hecubafb_dpy_deferred_io,
};
```

The delay is the minimum delay between when the page_mkwrite trigger occurs and when the deferred_io callback is called. The deferred_io callback is explained below.

2. Setup your deferred IO callback. Eg:

```
static void hecubafb_dpy_deferred_io(struct fb_info *info,
                                     struct list_head *pagelist)
```

The deferred_io callback is where you would perform all your IO to the display device. You receive the pagelist which is the list of pages that were written to during the delay. You must not modify this list. This callback is called from a workqueue.

3. Call init:

```
info->fbdefio = &hecubafb_defio;
fb_deferred_io_init(info);
```

4. Call cleanup:

```
fb_deferred_io_cleanup(info);
```

CHAPTER

SEVEN

WHAT IS EFIFB?

This is a generic EFI platform driver for systems with UEFI firmware. The system must be booted via the EFI stub for this to be usable. efifb supports both firmware with Graphics Output Protocol (GOP) displays as well as older systems with only Universal Graphics Adapter (UGA) displays.

7.1 Supported Hardware

- iMac 17”/20”
- Macbook
- Macbook Pro 15”/17”
- MacMini
- ARM/ARM64/X86 systems with UEFI firmware

7.2 How to use it?

For UGA displays, efifb does not have any kind of autodetection of your machine.

You have to add the following kernel parameters in your elilo.conf:

```
Macbook :
        video=efifb:macbook
MacMini :
        video=efifb:mini
Macbook Pro 15", iMac 17" :
        video=efifb:i17
Macbook Pro 17", iMac 20" :
        video=efifb:i20
```

For GOP displays, efifb can autodetect the display's resolution and framebuffer address, so these should work out of the box without any special parameters.

Accepted options:

nowc	Don't map the framebuffer write combined. This can be used to workaround side-effects and slowdowns on other CPU cores when large amounts of console data are written.

Options for GOP displays:

mode=n
: The EFI stub will set the mode of the display to mode number n if possible.

<xres>x<yres>[-(rgb|bgr|<bpp>)]
: The EFI stub will search for a display mode that matches the specified horizontal and vertical resolution, and optionally bit depth, and set the mode of the display to it if one is found. The bit depth can either ”rgb” or ”bgr” to match specifically those pixel formats, or a number for a mode with matching bits per pixel.

auto
: The EFI stub will choose the mode with the highest resolution (product of horizontal and vertical resolution). If there are multiple modes with the highest resolution, it will choose one with the highest color depth.

list
: The EFI stub will list out all the display modes that are available. A specific mode can then be chosen using one of the above options for the next boot.

Edgar Hucek <gimli@dark-green.com>

CHAPTER

EIGHT

DRIVER FOR EP93XX LCD CONTROLLER

The EP93xx LCD controller can drive both standard desktop monitors and embedded LCD displays. If you have a standard desktop monitor then you can use the standard Linux video mode database. In your board file:

```
static struct ep93xxfb_mach_info some_board_fb_info = {
        .num_modes      = EP93XXFB_USE_MODEDB,
        .bpp            = 16,
};
```

If you have an embedded LCD display then you need to define a video mode for it as follows:

```
static struct fb_videomode some_board_video_modes[] = {
        {
                .name           = "some_lcd_name",
                /* Pixel clock, porches, etc */
        },
};
```

Note that the pixel clock value is in pico-seconds. You can use the KHZ2PICOS macro to convert the pixel clock value. Most other values are in pixel clocks. See *The Frame Buffer Device* for further details.

The ep93xxfb_mach_info structure for your board should look like the following:

```
static struct ep93xxfb_mach_info some_board_fb_info = {
        .num_modes      = ARRAY_SIZE(some_board_video_modes),
        .modes          = some_board_video_modes,
        .default_mode   = &some_board_video_modes[0],
        .bpp            = 16,
};
```

The framebuffer device can be registered by adding the following to your board initialisation function:

```
ep93xx_register_fb(&some_board_fb_info);
```

CHAPTER

NINE

VIDEO ATTRIBUTE FLAGS

The ep93xxfb_mach_info structure has a flags field which can be used to configure the controller. The video attributes flags are fully documented in section 7 of the EP93xx users' guide. The following flags are available:

EP93XXFB_PCLK_FALLING	Clock data on the falling edge of the pixel clock. The default is to clock data on the rising edge.
EP93XXFB_SYNC_BLANK_HIGH	Blank signal is active high. By default the blank signal is active low.
EP93XXFB_SYNC_HORIZ_HIGH	Horizontal sync is active high. By default the horizontal sync is active low.
EP93XXFB_SYNC_VERT_HIGH	Vertical sync is active high. By default the vertical sync is active high.

The physical address of the framebuffer can be controlled using the following flags:

EP93XXFB_USE_SDCSN0	Use SDCSn[0] for the framebuffer. This is the default setting.
EP93XXFB_USE_SDCSN1	Use SDCSn[1] for the framebuffer.
EP93XXFB_USE_SDCSN2	Use SDCSn[2] for the framebuffer.
EP93XXFB_USE_SDCSN3	Use SDCSn[3] for the framebuffer.

CHAPTER

TEN

PLATFORM CALLBACKS

The EP93xx framebuffer driver supports three optional platform callbacks: setup, teardown and blank. The setup and teardown functions are called when the framebuffer driver is installed and removed respectively. The blank function is called whenever the display is blanked or unblanked.

The setup and teardown devices pass the platform_device structure as an argument. The fb_info and ep93xxfb_mach_info structures can be obtained as follows:

```
static int some_board_fb_setup(struct platform_device *pdev)
{
        struct ep93xxfb_mach_info *mach_info = pdev->dev.platform_data;
        struct fb_info *fb_info = platform_get_drvdata(pdev);

        /* Board specific framebuffer setup */
}
```

CHAPTER

ELEVEN

SETTING THE VIDEO MODE

The video mode is set using the following syntax:

```
video=XRESxYRES[-BPP][@REFRESH]
```

If the EP93xx video driver is built-in then the video mode is set on the Linux kernel command line, for example:

```
video=ep93xx-fb:800x600-16@60
```

If the EP93xx video driver is built as a module then the video mode is set when the module is installed:

```
modprobe ep93xx-fb video=320x240
```

CHAPTER

TWELVE

SCREENPAGE BUG

At least on the EP9315 there is a silicon bug which causes bit 27 of the VIDSCRNPAGE (framebuffer physical offset) to be tied low. There is an unofficial errata for this bug at:

```
https://marc.info/?l=linux-arm-kernel&m=110061245502000&w=2
```

By default the EP93xx framebuffer driver checks if the allocated physical address has bit 27 set. If it does, then the memory is freed and an error is returned. The check can be disabled by adding the following option when loading the driver:

```
ep93xx-fb.check_screenpage_bug=0
```

In some cases it may be possible to reconfigure your SDRAM layout to avoid this bug. See section 13 of the EP93xx users' guide for details.

CHAPTER

THIRTEEN

THE FRAMEBUFFER CONSOLE

The framebuffer console (fbcon), as its name implies, is a text console running on top of the framebuffer device. It has the functionality of any standard text console driver, such as the VGA console, with the added features that can be attributed to the graphical nature of the framebuffer.

In the x86 architecture, the framebuffer console is optional, and some even treat it as a toy. For other architectures, it is the only available display device, text or graphical.

What are the features of fbcon? The framebuffer console supports high resolutions, varying font types, display rotation, primitive multihead, etc. Theoretically, multi-colored fonts, blending, aliasing, and any feature made available by the underlying graphics card are also possible.

13.1 A. Configuration

The framebuffer console can be enabled by using your favorite kernel configuration tool. It is under Device Drivers->Graphics Support-> Console display driver support->Framebuffer Console Support. Select 'y' to compile support statically or 'm' for module support. The module will be fbcon.

In order for fbcon to activate, at least one framebuffer driver is required, so choose from any of the numerous drivers available. For x86 systems, they almost universally have VGA cards, so vga16fb and vesafb will always be available. However, using a chipset-specific driver will give you more speed and features, such as the ability to change the video mode dynamically.

To display the penguin logo, choose any logo available in Graphics support->Bootup logo.

Also, you will need to select at least one compiled-in font, but if you don't do anything, the kernel configuration tool will select one for you, usually an 8x16 font.

GOTCHA: A common bug report is enabling the framebuffer without enabling the framebuffer console. Depending on the driver, you may get a blanked or garbled display, but the system still boots to completion. If you are fortunate to have a driver that does not alter the graphics chip, then you will still get a VGA console.

13.2 B. Loading

Possible scenarios:

1. Driver and fbcon are compiled statically

   Usually, fbcon will automatically take over your console. The notable exception is vesafb. It needs to be explicitly activated with the vga= boot option parameter.

2. Driver is compiled statically, fbcon is compiled as a module

   Depending on the driver, you either get a standard console, or a garbled display, as mentioned above. To get a framebuffer console, do a 'modprobe fbcon'.

3. Driver is compiled as a module, fbcon is compiled statically

   You get your standard console. Once the driver is loaded with 'modprobe xxxfb', fbcon automatically takes over the console with the possible exception of using the fbcon=map:n option. See below.

4. Driver and fbcon are compiled as a module.

   You can load them in any order. Once both are loaded, fbcon will take over the console.

C. Boot options

   The framebuffer console has several, largely unknown, boot options that can change its behavior.

1. fbcon=font:<name>

   Select the initial font to use. The value 'name' can be any of the compiled-in fonts: 10x18, 6x10, 6x8, 7x14, Acorn8x8, MINI4x6, PEARL8x8, ProFont6x11, SUN12x22, SUN8x16, TER16x32, VGA8x16, VGA8x8.

   Note, not all drivers can handle font with widths not divisible by 8, such as vga16fb.

2. fbcon=map:<0123>

   This is an interesting option. It tells which driver gets mapped to which console. The value '0123' is a sequence that gets repeated until the total length is 64 which is the number of consoles available. In the above example, it is expanded to 012301230123... and the mapping will be:

   ```
   tty | 1 2 3 4 5 6 7 8 9 ...
   fb  | 0 1 2 3 0 1 2 3 0 ...

   ('cat /proc/fb' should tell you what the fb numbers are)
   ```

   One side effect that may be useful is using a map value that exceeds the number of loaded fb drivers. For example, if only one driver is available, fb0, adding fbcon=map:1 tells fbcon not to take over the console.

   Later on, when you want to map the console the to the framebuffer device, you can use the con2fbmap utility.

3. fbcon=vc:<n1>-<n2>

This option tells fbcon to take over only a range of consoles as specified by the values 'n1' and 'n2'. The rest of the consoles outside the given range will still be controlled by the standard console driver.

NOTE: For x86 machines, the standard console is the VGA console which is typically located on the same video card. Thus, the consoles that are controlled by the VGA console will be garbled.

4. fbcon=rotate:<n>

   This option changes the orientation angle of the console display. The value 'n' accepts the following:

   - 0 - normal orientation (0 degree)
   - 1 - clockwise orientation (90 degrees)
   - 2 - upside down orientation (180 degrees)
   - 3 - counterclockwise orientation (270 degrees)

   The angle can be changed anytime afterwards by 'echoing' the same numbers to any one of the 2 attributes found in /sys/class/graphics/fbcon:

   - rotate - rotate the display of the active console
   - rotate_all - rotate the display of all consoles

   Console rotation will only become available if Framebuffer Console Rotation support is compiled in your kernel.

   NOTE: This is purely console rotation. Any other applications that use the framebuffer will remain at their 'normal' orientation. Actually, the underlying fb driver is totally ignorant of console rotation.

5. fbcon=margin:<color>

   This option specifies the color of the margins. The margins are the leftover area at the right and the bottom of the screen that are not used by text. By default, this area will be black. The 'color' value is an integer number that depends on the framebuffer driver being used.

6. fbcon=nodefer

   If the kernel is compiled with deferred fbcon takeover support, normally the framebuffer contents, left in place by the firmware/bootloader, will be preserved until there actually is some text is output to the console. This option causes fbcon to bind immediately to the fbdev device.

7. fbcon=logo-pos:<location>

   The only possible 'location' is 'center' (without quotes), and when given, the bootup logo is moved from the default top-left corner location to the center of the framebuffer. If more than one logo is displayed due to multiple CPUs, the collected line of logos is moved as a whole.

8. fbcon=logo-count:<n>

   The value 'n' overrides the number of bootup logos. 0 disables the logo, and -1 gives the default which is the number of online CPUs.

C. Attaching, Detaching and Unloading

Before going on to how to attach, detach and unload the framebuffer console, an illustration of the dependencies may help.

The console layer, as with most subsystems, needs a driver that interfaces with the hardware. Thus, in a VGA console:

```
console ---> VGA driver ---> hardware.
```

Assuming the VGA driver can be unloaded, one must first unbind the VGA driver from the console layer before unloading the driver. The VGA driver cannot be unloaded if it is still bound to the console layer. (See Documentation/driver-api/console.rst for more information).

This is more complicated in the case of the framebuffer console (fbcon), because fbcon is an intermediate layer between the console and the drivers:

```
console ---> fbcon ---> fbdev drivers ---> hardware
```

The fbdev drivers cannot be unloaded if bound to fbcon, and fbcon cannot be unloaded if it's bound to the console layer.

So to unload the fbdev drivers, one must first unbind fbcon from the console, then unbind the fbdev drivers from fbcon. Fortunately, unbinding fbcon from the console layer will automatically unbind framebuffer drivers from fbcon. Thus, there is no need to explicitly unbind the fbdev drivers from fbcon.

So, how do we unbind fbcon from the console? Part of the answer is in Documentation/driver-api/console.rst. To summarize:

Echo a value to the bind file that represents the framebuffer console driver. So assuming vtcon1 represents fbcon, then:

```
echo 1 > /sys/class/vtconsole/vtcon1/bind - attach framebuffer console to
                                            console layer
echo 0 > /sys/class/vtconsole/vtcon1/bind - detach framebuffer console from
                                            console layer
```

If fbcon is detached from the console layer, your boot console driver (which is usually VGA text mode) will take over. A few drivers (rivafb and i810fb) will restore VGA text mode for you. With the rest, before detaching fbcon, you must take a few additional steps to make sure that your VGA text mode is restored properly. The following is one of the several methods that you can do:

1. Download or install vbetool. This utility is included with most distributions nowadays, and is usually part of the suspend/resume tool.
2. In your kernel configuration, ensure that CONFIG_FRAMEBUFFER_CONSOLE is set to 'y' or 'm'. Enable one or more of your favorite framebuffer drivers.
3. Boot into text mode and as root run:

   ```
   vbetool vbestate save > <vga state file>
   ```

   The above command saves the register contents of your graphics hardware to <vga state file>. You need to do this step only once as the state file can be reused.
4. If fbcon is compiled as a module, load fbcon by doing:

```
modprobe fbcon
```

5. Now to detach fbcon:

```
vbetool vbestate restore < <vga state file> && \
echo 0 > /sys/class/vtconsole/vtcon1/bind
```

6. That's it, you're back to VGA mode. And if you compiled fbcon as a module, you can unload it by 'rmmod fbcon'.

7. To reattach fbcon:

```
echo 1 > /sys/class/vtconsole/vtcon1/bind
```

8. Once fbcon is unbound, all drivers registered to the system will also become unbound. This means that fbcon and individual framebuffer drivers can be unloaded or reloaded at will. Reloading the drivers or fbcon will automatically bind the console, fbcon and the drivers together. Unloading all the drivers without unloading fbcon will make it impossible for the console to bind fbcon.

13.3 Notes for vesafb users:

Unfortunately, if your bootline includes a vga=xxx parameter that sets the hardware in graphics mode, such as when loading vesafb, vgacon will not load. Instead, vgacon will replace the default boot console with dummycon, and you won't get any display after detaching fbcon. Your machine is still alive, so you can reattach vesafb. However, to reattach vesafb, you need to do one of the following:

Variation 1:

a. Before detaching fbcon, do:

```
vbetool vbemode save > <vesa state file> # do once for each vesafb mode,
                                         # the file can be reused
```

b. Detach fbcon as in step 5.

c. Attach fbcon:

```
vbetool vbestate restore < <vesa state file> && \
echo 1 > /sys/class/vtconsole/vtcon1/bind
```

Variation 2:

a. Before detaching fbcon, do:

```
echo <ID> > /sys/class/tty/console/bind

vbetool vbemode get
```

b. Take note of the mode number

b. Detach fbcon as in step 5.

c. Attach fbcon:

```
vbetool vbemode set <mode number> && \
echo 1 > /sys/class/vtconsole/vtcon1/bind
```

13.4 Samples:

Here are 2 sample bash scripts that you can use to bind or unbind the framebuffer console driver if you are on an X86 box:

```
#!/bin/bash
# Unbind fbcon

# Change this to where your actual vgastate file is located
# Or Use VGASTATE=$1 to indicate the state file at runtime
VGASTATE=/tmp/vgastate

# path to vbetool
VBETOOL=/usr/local/bin


for (( i = 0; i < 16; i++))
do
  if test -x /sys/class/vtconsole/vtcon$i; then
      if [ `cat /sys/class/vtconsole/vtcon$i/name | grep -c "frame buffer"` \
           = 1 ]; then
          if test -x $VBETOOL/vbetool; then
             echo Unbinding vtcon$i
             $VBETOOL/vbetool vbestate restore < $VGASTATE
             echo 0 > /sys/class/vtconsole/vtcon$i/bind
          fi
      fi
  fi
done
```

```
#!/bin/bash
# Bind fbcon

for (( i = 0; i < 16; i++))
do
  if test -x /sys/class/vtconsole/vtcon$i; then
      if [ `cat /sys/class/vtconsole/vtcon$i/name | grep -c "frame buffer"` \
           = 1 ]; then
        echo Unbinding vtcon$i
        echo 1 > /sys/class/vtconsole/vtcon$i/bind
      fi
  fi
done
```

Antonino Daplas <adaplas@pol.net>

CHAPTER

FOURTEEN

THE FRAME BUFFER DEVICE

Last revised: May 10, 2001

14.1 0. Introduction

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 doesn't need to know anything about the low-level (hardware register) stuff.

The device is accessed through special device nodes, usually located in the /dev directory, i.e. /dev/fb*.

14.2 1. User's View of /dev/fb*

From the user's point of view, the frame buffer device looks just like any other device in /dev. It's a character device using major 29; the minor specifies the frame buffer number.

By convention, the following device nodes are used (numbers indicate the device minor numbers):

```
 0 = /dev/fb0      First frame buffer
 1 = /dev/fb1      Second frame buffer
     ...
31 = /dev/fb31     32nd frame buffer
```

For backwards compatibility, you may want to create the following symbolic links:

```
/dev/fb0current -> fb0
/dev/fb1current -> fb1
```

and so on...

The frame buffer devices are also *normal* memory devices, this means, you can read and write their contents. You can, for example, make a screen snapshot by:

```
cp /dev/fb0 myfile
```

There also can be more than one frame buffer at a time, e.g. if you have a graphics card in addition to the built-in hardware. The corresponding frame buffer devices (/dev/fb0 and /dev/fb1 etc.) work independently.

Application software that uses the frame buffer device (e.g. the X server) will use /dev/fb0 by default (older software uses /dev/fb0current). You can specify an alternative frame buffer device by setting the environment variable $FRAMEBUFFER to the path name of a frame buffer device, e.g. (for sh/bash users):

```
export FRAMEBUFFER=/dev/fb1
```

or (for csh users):

```
setenv FRAMEBUFFER /dev/fb1
```

After this the X server will use the second frame buffer.

14.3 2. Programmer's View of /dev/fb*

As you already know, a frame buffer device is a memory device like /dev/mem and it has the same features. You can read it, write it, seek to some location in it and mmap() it (the main usage). The difference is just that the memory that appears in the special file is not the whole memory, but the frame buffer of some video hardware.

/dev/fb* also allows several ioctls on it, by which lots of information about the hardware can be queried and set. The color map handling works via ioctls, too. Look into <linux/fb.h> for more information on what ioctls exist and on which data structures they work. Here's just a brief overview:

- You can request unchangeable information about the hardware, like name, organization of the screen memory (planes, packed pixels, ...) and address and length of the screen memory.
- You can request and change variable information about the hardware, like visible and virtual geometry, depth, color map format, timing, and so on. If you try to change that information, the driver maybe will round up some values to meet the hardware's capabilities (or return EINVAL if that isn't possible).
- You can get and set parts of the color map. Communication is done with 16 bits per color part (red, green, blue, transparency) to support all existing hardware. The driver does all the computations needed to apply it to the hardware (round it down to less bits, maybe throw away transparency).

All this hardware abstraction makes the implementation of application programs easier and more portable. E.g. the X server works completely on /dev/fb* and thus doesn't need to know, for example, how the color registers of the concrete hardware are organized. XF68_FBDev is a general X server for bitmapped, unaccelerated video hardware. The only thing that has to be built into application programs is the screen organization (bitplanes or chunky pixels etc.), because it works on the frame buffer image data directly.

For the future it is planned that frame buffer drivers for graphics cards and the like can be implemented as kernel modules that are loaded at runtime. Such a driver just has to call register_framebuffer() and supply some functions. Writing and distributing such drivers independently from the kernel will save much trouble...

14.4 3. Frame Buffer Resolution Maintenance

Frame buffer resolutions are maintained using the utility *fbset*. It can change the video mode properties of a frame buffer device. Its main usage is to change the current video mode, e.g. during boot up in one of your */etc/rc.** or */etc/init.d/** files.

Fbset uses a video mode database stored in a configuration file, so you can easily add your own modes and refer to them with a simple identifier.

14.5 4. The X Server

The X server (XF68_FBDev) is the most notable application program for the frame buffer device. Starting with XFree86 release 3.2, the X server is part of XFree86 and has 2 modes:

- If the *Display* subsection for the *fbdev* driver in the /etc/XF86Config file contains a:

  ```
  Modes "default"
  ```

  line, the X server will use the scheme discussed above, i.e. it will start up in the resolution determined by /dev/fb0 (or $FRAMEBUFFER, if set). You still have to specify the color depth (using the Depth keyword) and virtual resolution (using the Virtual keyword) though. This is the default for the configuration file supplied with XFree86. It's the most simple configuration, but it has some limitations.

- Therefore it's also possible to specify resolutions in the /etc/XF86Config file. This allows for on-the-fly resolution switching while retaining the same virtual desktop size. The frame buffer device that's used is still /dev/fb0current (or $FRAMEBUFFER), but the available resolutions are defined by /etc/XF86Config now. The disadvantage is that you have to specify the timings in a different format (but *fbset -x* may help).

To tune a video mode, you can use fbset or xvidtune. Note that xvidtune doesn't work 100% with XF68_FBDev: the reported clock values are always incorrect.

14.6 5. Video Mode Timings

A monitor draws an image on the screen by using an electron beam (3 electron beams for color models, 1 electron beam for monochrome monitors). The front of the screen is covered by a pattern of colored phosphors (pixels). If a phosphor is hit by an electron, it emits a photon and thus becomes visible.

The electron beam draws horizontal lines (scanlines) from left to right, and from the top to the bottom of the screen. By modifying the intensity of the electron beam, pixels with various colors and intensities can be shown.

After each scanline the electron beam has to move back to the left side of the screen and to the next line: this is called the horizontal retrace. After the whole screen (frame) was painted, the beam moves back to the upper left corner: this is called the vertical retrace. During both the horizontal and vertical retrace, the electron beam is turned off (blanked).

The speed at which the electron beam paints the pixels is determined by the dotclock in the graphics board. For a dotclock of e.g. 28.37516 MHz (millions of cycles per second), each pixel is 35242 ps (picoseconds) long:

```
1/(28.37516E6 Hz) = 35.242E-9 s
```

If the screen resolution is 640x480, it will take:

```
640*35.242E-9 s = 22.555E-6 s
```

to paint the 640 (xres) pixels on one scanline. But the horizontal retrace also takes time (e.g. 272 *pixels*), so a full scanline takes:

```
(640+272)*35.242E-9 s = 32.141E-6 s
```

We'll say that the horizontal scanrate is about 31 kHz:

```
1/(32.141E-6 s) = 31.113E3 Hz
```

A full screen counts 480 (yres) lines, but we have to consider the vertical retrace too (e.g. 49 *lines*). So a full screen will take:

```
(480+49)*32.141E-6 s = 17.002E-3 s
```

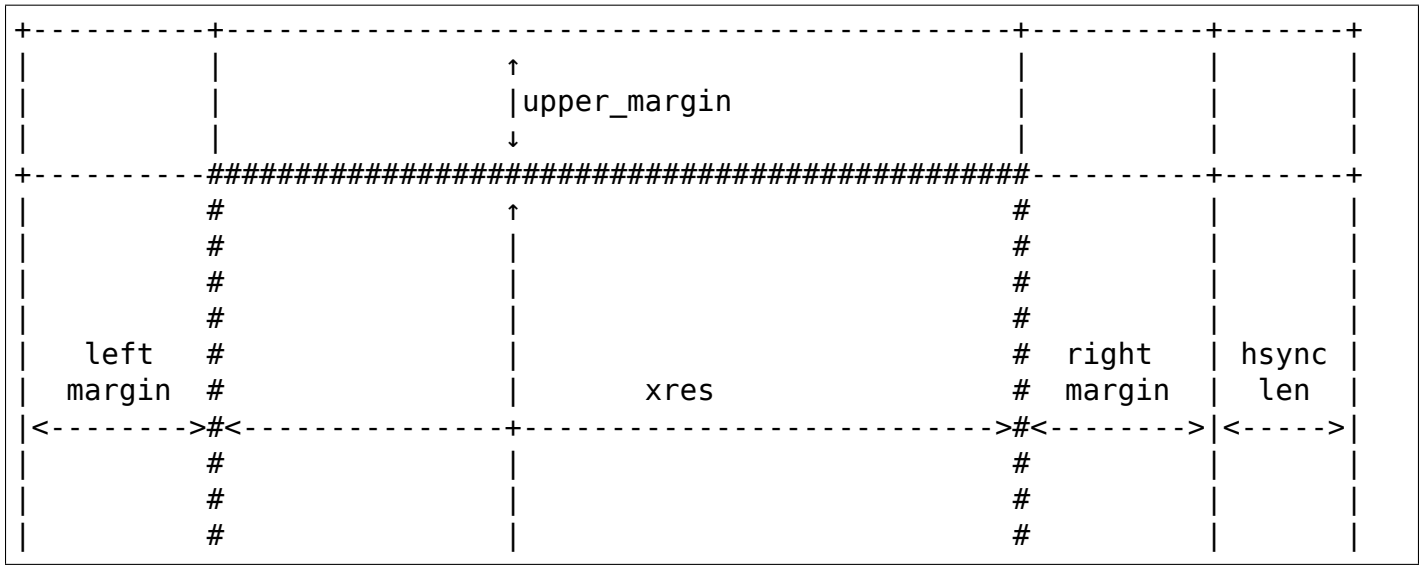
The vertical scanrate is about 59 Hz:

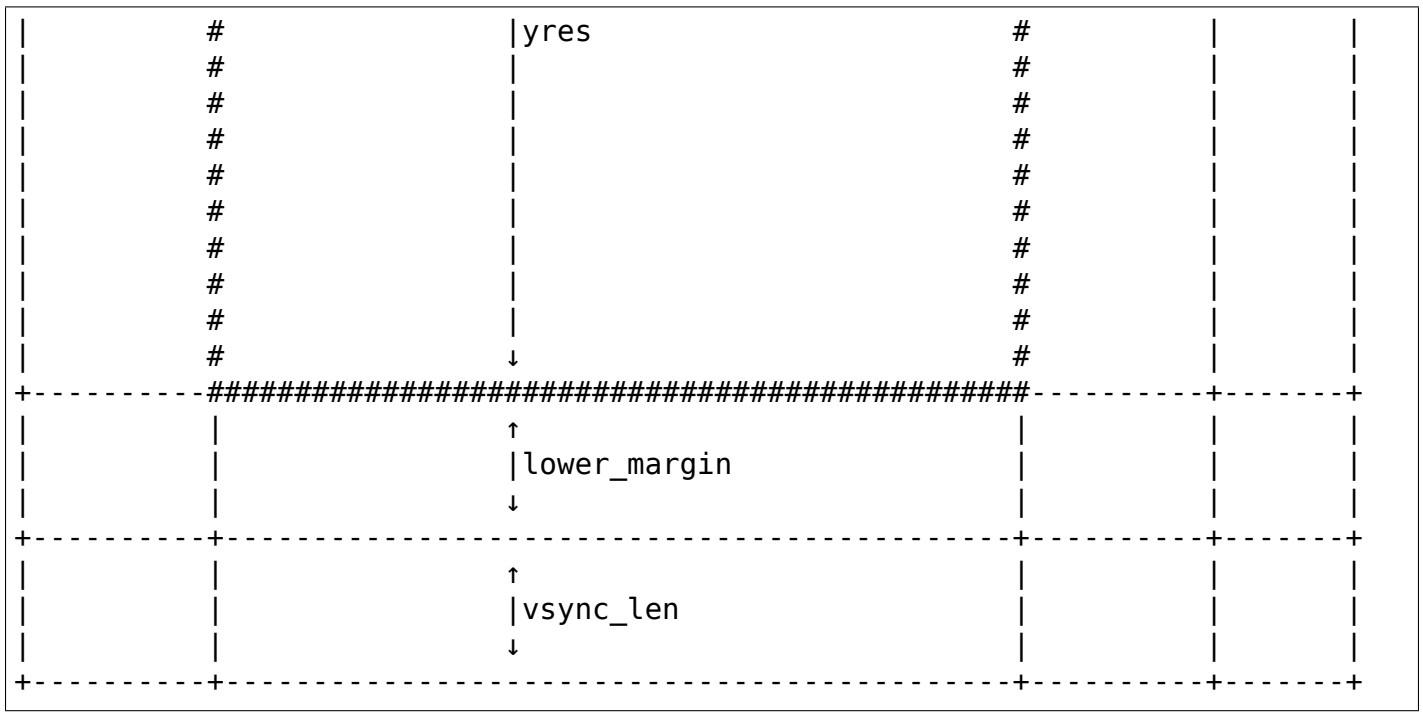
```
1/(17.002E-3 s) = 58.815 Hz
```

This means the screen data is refreshed about 59 times per second. To have a stable picture without visible flicker, VESA recommends a vertical scanrate of at least 72 Hz. But the perceived flicker is very human dependent: some people can use 50 Hz without any trouble, while I'll notice if it's less than 80 Hz.

Since the monitor doesn't know when a new scanline starts, the graphics board will supply a synchronization pulse (horizontal sync or hsync) for each scanline. Similarly it supplies a synchronization pulse (vertical sync or vsync) for each new frame. The position of the image on the screen is influenced by the moments at which the synchronization pulses occur.

The following picture summarizes all timings. The horizontal retrace time is the sum of the left margin, the right margin and the hsync length, while the vertical retrace time is the sum of the upper margin, the lower margin and the vsync length:





The frame buffer device expects all horizontal timings in number of dotclocks (in picoseconds, 1E-12 s), and vertical timings in number of scanlines.

14.7 6. Converting XFree86 timing values info frame buffer device timings

An XFree86 mode line consists of the following fields:

```
"800x600"     50      800  856  976 1040    600  637  643  666
< name >     DCF       HR  SH1  SH2  HFL     VR  SV1  SV2  VFL
```

The frame buffer device uses the following fields:

- pixclock: pixel clock in ps (pico seconds)
- left_margin: time from sync to picture
- right_margin: time from picture to sync
- upper_margin: time from sync to picture
- lower_margin: time from picture to sync
- hsync_len: length of horizontal sync
- vsync_len: length of vertical sync

1) Pixelclock:

   xfree: in MHz

   fb: in picoseconds (ps)

   pixclock = 1000000 / DCF

2) horizontal timings:

   left_margin = HFL - SH2

   right_margin = SH1 - HR

   hsync_len = SH2 - SH1

3) vertical timings:

   upper_margin = VFL - SV2

   lower_margin = SV1 - VR

   vsync_len = SV2 - SV1

Good examples for VESA timings can be found in the XFree86 source tree, under "xc/programs/Xserver/hw/xfree86/doc/modeDB.txt".

14.8 7. References

For more specific information about the frame buffer device and its applications, please refer to the Linux-fbdev website:

http://linux-fbdev.sourceforge.net/

and to the following documentation:

- The manual pages for fbset: fbset(8), fb.modes(5)
- The manual pages for XFree86: XF68_FBDev(1), XF86Config(4/5)
- The mighty kernel sources:
  - linux/drivers/video/
  - linux/include/linux/fb.h
  - linux/include/video/

14.9 8. Mailing list

There is a frame buffer device related mailing list at kernel.org: linux-fbdev@vger.kernel.org.

Point your web browser to http://sourceforge.net/projects/linux-fbdev/ for subscription information and archive browsing.

14.10 9. Downloading

All necessary files can be found at

ftp://ftp.uni-erlangen.de/pub/Linux/LOCAL/680x0/

and on its mirrors.

The latest version of fbset can be found at

http://www.linux-fbdev.org/

14.11 10. Credits

This readme was written by Geert Uytterhoeven, partly based on the original *X-framebuffer.README* by Roman Hodek and Martin Schaller. Section 6 was provided by Frank Neumann.

The frame buffer device abstraction was designed by Martin Schaller.

CHAPTER

FIFTEEN

WHAT IS GXFB?

This is a graphics framebuffer driver for AMD Geode GX2 based processors.

Advantages:

- No need to use AMD's VSA code (or other VESA emulation layer) in the BIOS.
- It provides a nice large console (128 cols + 48 lines with 1024x768) without using tiny, unreadable fonts.
- You can run XF68_FBDev on top of /dev/fb0
- Most important: boot logo :-)

Disadvantages:

- graphic mode is slower than text mode...

15.1 How to use it?

Switching modes is done using gxfb.mode_option=<resolution>... boot parameter or using *fbset* program.

See *modedb default video mode support* for more information on modedb resolutions.

15.2 X11

XF68_FBDev should generally work fine, but it is non-accelerated.

15.3 Configuration

You can pass kernel command line options to gxfb with gxfb.<option>. For example, gxfb.mode_option=800x600@75. Accepted options:

mode_option	specify the video mode. Of the form <x>x<y>[-<bpp>][@<refresh>]
vram	size of video ram (normally auto-detected)
vt_switch	enable vt switching during suspend/resume. The vt switch is slow, but harmless.

Andres Salomon <dilinger@debian.org>

CHAPTER

SIXTEEN

INTEL 810/815 FRAMEBUFFER DRIVER

Tony Daplas <adaplas@pol.net>

http://i810fb.sourceforge.net

March 17, 2002

First Released: July 2001 Last Update: September 12, 2005

16.1 A. Introduction

This is a framebuffer driver for various Intel 810/815 compatible graphics devices. These include:

- Intel 810
- Intel 810E
- Intel 810-DC100
- Intel 815 Internal graphics only, 100Mhz FSB
- Intel 815 Internal graphics only
- Intel 815 Internal graphics and AGP

16.2 B. Features

- Choice of using Discrete Video Timings, VESA Generalized Timing Formula, or a framebuffer specific database to set the video mode
- Supports a variable range of horizontal and vertical resolution and vertical refresh rates if the VESA Generalized Timing Formula is enabled.
- Supports color depths of 8, 16, 24 and 32 bits per pixel
- Supports pseudocolor, directcolor, or truecolor visuals
- Full and optimized hardware acceleration at 8, 16 and 24 bpp
- Robust video state save and restore
- MTRR support
- Utilizes user-entered monitor specifications to automatically calculate required video mode parameters.

- Can concurrently run with xfree86 running with native i810 drivers
- Hardware Cursor Support
- Supports EDID probing either by DDC/I2C or through the BIOS

16.3 C. List of available options

a. **”video=i810fb”**
   enables the i810 driver

   Recommendation: required

b. **”xres:<value>”**
   select horizontal resolution in pixels. (This parameter will be ignored if 'mode_option' is specified. See 'o' below).

   Recommendation: user preference (default = 640)

c. **”yres:<value>”**
   select vertical resolution in scanlines. If Discrete Video Timings is enabled, this will be ignored and computed as 3*xres/4. (This parameter will be ignored if 'mode_option' is specified. See 'o' below)

   Recommendation: user preference (default = 480)

d. **”vyres:<value>”**
   select virtual vertical resolution in scanlines. If (0) or none is specified, this will be computed against maximum available memory.

   Recommendation: do not set (default = 480)

e. **”vram:<value>”**
   select amount of system RAM in MB to allocate for the video memory

   Recommendation: 1 - 4 MB. (default = 4)

f. **”bpp:<value>”**
   select desired pixel depth

   Recommendation: 8 (default = 8)

g. **”hsync1/hsync2:<value>”**
   select the minimum and maximum Horizontal Sync Frequency of the monitor in kHz. If using a fixed frequency monitor, hsync1 must be equal to hsync2. If EDID probing is successful, these will be ignored and values will be taken from the EDID block.

   Recommendation: check monitor manual for correct values (default = 29/30)

h. **”vsync1/vsync2:<value>”**
   select the minimum and maximum Vertical Sync Frequency of the monitor in Hz. You can also use this option to lock your monitor's refresh rate. If EDID probing is successful, these will be ignored and values will be taken from the EDID block.

   Recommendation: check monitor manual for correct values (default = 60/60)

   IMPORTANT: If you need to clamp your timings, try to give some leeway for computational errors (over/underflows). Example: if using vsync1/vsync2 = 60/60, make sure hsync1/hsync2 has at least a 1 unit difference, and vice versa.

i. **”voffset:<value>”**
   select at what offset in MB of the logical memory to allocate the framebuffer memory. The intent is to avoid the memory blocks used by standard graphics applications (XFree86). The default offset (16 MB for a 64 MB aperture, 8 MB for a 32 MB aperture) will avoid XFree86's usage and allows up to 7 MB/15 MB of framebuffer memory. Depending on your usage, adjust the value up or down (0 for maximum usage, 31/63 MB for the least amount). Note, an arbitrary setting may conflict with XFree86.

   Recommendation: do not set (default = 8 or 16 MB)

j. **”accel”**
   enable text acceleration. This can be enabled/reenabled anytime by using 'fbset -accel true/false'.

   Recommendation: enable (default = not set)

k. **”mtrr”**
   enable MTRR. This allows data transfers to the framebuffer memory to occur in bursts which can significantly increase performance. Not very helpful with the i810/i815 because of 'shared memory'.

   Recommendation: do not set (default = not set)

l. **”extvga”**
   if specified, secondary/external VGA output will always be enabled. Useful if the BIOS turns off the VGA port when no monitor is attached. The external VGA monitor can then be attached without rebooting.

   Recommendation: do not set (default = not set)

m. **”sync”**
   Forces the hardware engine to do a ”sync” or wait for the hardware to finish before starting another instruction. This will produce a more stable setup, but will be slower.

   Recommendation: do not set (default = not set)

n. **”dcolor”**
   Use directcolor visual instead of truecolor for pixel depths greater than 8 bpp. Useful for color tuning, such as gamma control.

   Recommendation: do not set (default = not set)

o. **<xres>x<yres>[-<bpp>][@<refresh>]**
   The driver will now accept specification of boot mode option. If this is specified, the options 'xres' and 'yres' will be ignored. See *modedb default video mode support* for usage.

16.4 D. Kernel booting

Separate each option/option-pair by commas (,) and the option from its value with a colon (:) as in the following:

```
video=i810fb:option1,option2:value2
```

16.4.1 Sample Usage

In /etc/lilo.conf, add the line:

```
append="video=i810fb:vram:2,xres:1024,yres:768,bpp:8,hsync1:30,hsync2:55, \
        vsync1:50,vsync2:85,accel,mtrr"
```

This will initialize the framebuffer to 1024x768 at 8bpp. The framebuffer will use 2 MB of System RAM. MTRR support will be enabled. The refresh rate will be computed based on the hsync1/hsync2 and vsync1/vsync2 values.

IMPORTANT:
You must include hsync1, hsync2, vsync1 and vsync2 to enable video modes better than 640x480 at 60Hz. HOWEVER, if your chipset/display combination supports I2C and has an EDID block, you can safely exclude hsync1, hsync2, vsync1 and vsync2 parameters. These parameters will be taken from the EDID block.

16.5 E. Module options

The module parameters are essentially similar to the kernel parameters. The main difference is that you need to include a Boolean value (1 for TRUE, and 0 for FALSE) for those options which don't need a value.

Example, to enable MTRR, include ”mtrr=1”.

16.5.1 Sample Usage

Using the same setup as described above, load the module like this:

```
modprobe i810fb vram=2 xres=1024 bpp=8 hsync1=30 hsync2=55 vsync1=50 \
         vsync2=85 accel=1 mtrr=1
```

Or just add the following to a configuration file in /etc/modprobe.d/:

```
options i810fb vram=2 xres=1024 bpp=16 hsync1=30 hsync2=55 vsync1=50 \
vsync2=85 accel=1 mtrr=1
```

and just do a:

```
modprobe i810fb
```

16.6 F. Setup

a. Do your usual method of configuring the kernel

   make menuconfig/xconfig/config

b. Under ”Code maturity level options” enable ”Prompt for development and/or incomplete code/drivers”.

c. Enable agpgart support for the Intel 810/815 on-board graphics. This is required. The option is under ”Character Devices”.

d. Under ”Graphics Support”, select ”Intel 810/815” either statically or as a module. Choose ”use VESA Generalized Timing Formula” if you need to maximize the capability of your display. To be on the safe side, you can leave this unselected.

e. If you want support for DDC/I2C probing (Plug and Play Displays), set 'Enable DDC Support' to 'y'. To make this option appear, set 'use VESA Generalized Timing Formula' to 'y'.

f. If you want a framebuffer console, enable it under ”Console Drivers”.

g. Compile your kernel.

h. Load the driver as described in sections D and E.

i. Try the DirectFB (http://www.directfb.org) + the i810 gfxdriver patch to see the chipset in action (or inaction :-).

16.7 G. Acknowledgment:

1. Geert Uytterhoeven - his excellent howto and the virtual framebuffer driver code made this possible.
2. Jeff Hartmann for his agpgart code.
3. The X developers. Insights were provided just by reading the XFree86 source code.
4. Intel(c). For this value-oriented chipset driver and for providing documentation.
5. Matt Sottek. His inputs and ideas helped in making some optimizations possible.

16.8 H. Home Page:

A more complete, and probably updated information is provided at http://i810fb.sourceforge.net.

Tony

CHAPTER

SEVENTEEN

FRAME BUFFER DEVICE INTERNALS

This is a first start for some documentation about frame buffer device internals.

Authors:

- Geert Uytterhoeven <geert@linux-m68k.org>, 21 July 1998
- James Simmons <jsimmons@user.sf.net>, Nov 26 2002

17.1 Structures used by the frame buffer device API

The following structures play a role in the game of frame buffer devices. They are defined in <linux/fb.h>.

1. Outside the kernel (user space)

- struct fb_fix_screeninfo

  Device independent unchangeable information about a frame buffer device and a specific video mode. This can be obtained using the FBIOGET_FSCREENINFO ioctl.

- struct fb_var_screeninfo

  Device independent changeable information about a frame buffer device and a specific video mode. This can be obtained using the FBIOGET_VSCREENINFO ioctl, and updated with the FBIOPUT_VSCREENINFO ioctl. If you want to pan the screen only, you can use the FBIOPAN_DISPLAY ioctl.

- struct fb_cmap

  Device independent colormap information. You can get and set the colormap using the FBIOGETCMAP and FBIOPUTCMAP ioctls.

2. Inside the kernel

- struct fb_info

  Generic information, API and low level information about a specific frame buffer device instance (slot number, board address, ...).

- struct *par*

  Device dependent information that uniquely defines the video mode for this particular piece of hardware.

17.2 Visuals used by the frame buffer device API

17.2.1 Monochrome (FB_VISUAL_MONO01 and FB_VISUAL_MONO10)

Each pixel is either black or white.

17.2.2 Pseudo color (FB_VISUAL_PSEUDOCOLOR and FB_VISUAL_STATIC_PSEUDOCOL

The whole pixel value is fed through a programmable lookup table that has one color (including red, green, and blue intensities) for each possible pixel value, and that color is displayed.

17.2.3 True color (FB_VISUAL_TRUECOLOR)

The pixel value is broken up into red, green, and blue fields.

17.2.4 Direct color (FB_VISUAL_DIRECTCOLOR)

The pixel value is broken up into red, green, and blue fields, each of which are looked up in separate red, green, and blue lookup tables.

17.2.5 Grayscale displays

Grayscale and static grayscale are special variants of pseudo color and static pseudo color, where the red, green and blue components are always equal to each other.

CHAPTER

EIGHTEEN

WHAT IS LXFB?

This is a graphics framebuffer driver for AMD Geode LX based processors.

Advantages:

- No need to use AMD's VSA code (or other VESA emulation layer) in the BIOS.
- It provides a nice large console (128 cols + 48 lines with 1024x768) without using tiny, unreadable fonts.
- You can run XF68_FBDev on top of /dev/fb0
- Most important: boot logo :-)

Disadvantages:

- graphic mode is slower than text mode...

18.1 How to use it?

Switching modes is done using lxfb.mode_option=<resolution>... boot parameter or using *fbset* program.

See *modedb default video mode support* for more information on modedb resolutions.

18.2 X11

XF68_FBDev should generally work fine, but it is non-accelerated.

18.3 Configuration

You can pass kernel command line options to lxfb with lxfb.<option>. For example, lxfb.mode_option=800x600@75. Accepted options:

mode_option	specify the video mode. Of the form <x>x<y>[-<bpp>][@<refresh>]
vram	size of video ram (normally auto-detected)
vt_switch	enable vt switching during suspend/resume. The vt switch is slow, but harmless.

Andres Salomon <dilinger@debian.org>

CHAPTER

NINETEEN

WHAT IS MATROXFB?

This is a driver for a graphic framebuffer for Matrox devices on Alpha, Intel and PPC boxes.

Advantages:

- It provides a nice large console (128 cols + 48 lines with 1024x768) without using tiny, unreadable fonts.
- You can run XF{68,86}_FBDev or XFree86 fbdev driver on top of /dev/fb0
- Most important: boot logo :-)

Disadvantages:

- graphic mode is slower than text mode... but you should not notice if you use same resolution as you used in textmode.

19.1 How to use it?

Switching modes is done using the video=matroxfb:vesa:... boot parameter or using *fbset* program.

If you want, for example, enable a resolution of 1280x1024x24bpp you should pass to the kernel this command line: "video=matroxfb:vesa:0x1BB".

You should compile in both vgacon (to boot if you remove you Matrox from box) and matroxfb (for graphics mode). You should not compile-in vesafb unless you have primary display on non-Matrox VBE2.0 device (see *What is vesafb?* for details).

Currently supported video modes are (through vesa:... interface, PowerMac has [as addon] compatibility code):

19.1.1 Graphic modes

bpp	640x400	640x480	768x576	800x600	960x720
4		0x12		0x102	
8	0x100	0x101	0x180	0x103	0x188
15		0x110	0x181	0x113	0x189
16		0x111	0x182	0x114	0x18A
24		0x1B2	0x184	0x1B5	0x18C
32		0x112	0x183	0x115	0x18B

19.1.2 Graphic modes (continued)

bpp	1024x768	1152x864	1280x1024	1408x1056	1600x1200
4	0x104		0x106		
8	0x105	0x190	0x107	0x198	0x11C
15	0x116	0x191	0x119	0x199	0x11D
16	0x117	0x192	0x11A	0x19A	0x11E
24	0x1B8	0x194	0x1BB	0x19C	0x1BF
32	0x118	0x193	0x11B	0x19B	

19.1.3 Text modes

text	640x400	640x480	1056x344	1056x400	1056x480
8x8	0x1C0	0x108	0x10A	0x10B	0x10C
8x16	2, 3, 7			0x109	

You can enter these number either hexadecimal (leading *0x*) or decimal (0x100 = 256). You can also use value + 512 to achieve compatibility with your old number passed to vesafb.

Non-listed number can be achieved by more complicated command-line, for example 1600x1200x32bpp can be specified by *video=matroxfb:vesa:0x11C,depth:32*.

19.2 X11

XF{68,86}_FBDev should work just fine, but it is non-accelerated. On non-intel architectures there are some glitches for 24bpp videomodes. 8, 16 and 32bpp works fine.

Running another (accelerated) X-Server like XF86_SVGA works too. But (at least) XFree servers have big troubles in multihead configurations (even on first head, not even talking about second). Running XFree86 4.x accelerated mga driver is possible, but you must not enable DRI - if you do, resolution and color depth of your X desktop must match resolution and color depths of your virtual consoles, otherwise X will corrupt accelerator settings.

19.3 SVGALib

Driver contains SVGALib compatibility code. It is turned on by choosing textual mode for console. You can do it at boot time by using videomode 2,3,7,0x108-0x10C or 0x1C0. At runtime, *fbset -depth 0* does this work. Unfortunately, after SVGALib application exits, screen contents is corrupted. Switching to another console and back fixes it. I hope that it is SVGALib's problem and not mine, but I'm not sure.

19.4 Configuration

You can pass kernel command line options to matroxfb with *video=matroxfb:option1,option2:value2,option3* (multiple options should be separated by comma, values are separated from options by *:*). Accepted options:

mem:X	size of memory (X can be in megabytes, kilobytes or bytes) You can only decrease value determined by driver because of it always probe for memory. Default is to use whole detected memory usable for on-screen display (i.e. max. 8 MB).
disabled	do not load driver; you can use also *off*, but *disabled* is here too.
enabled	load driver, if you have *video=matroxfb:disabled* in LILO configuration, you can override it by this (you cannot override *off*). It is default.
noaccel	do not use acceleration engine. It does not work on Alphas.
accel	use acceleration engine. It is default.
nopan	create initial consoles with vyres = yres, thus disabling virtual scrolling.
pan	create initial consoles as tall as possible (vyres = memory/vxres). It is default.
nopciretry	disable PCI retries. It is needed for some broken chipsets, it is autodetected for intel's 82437. In this case device does not comply to PCI 2.1 specs (it will not guarantee that every transaction terminate with success or retry in 32 PCLK).
pciretry	enable PCI retries. It is default, except for intel's 82437.
novga	disables VGA I/O ports. It is default if BIOS did not enable device. You should not use this option, some boards then do not restart without power off.
vga	preserve state of VGA I/O ports. It is default. Driver does not enable VGA I/O if BIOS did not it (it is not safe to enable it in most cases).
nobios	disables BIOS ROM. It is default if BIOS did not enable BIOS itself. You should not use this option, some boards then do not restart without power off.
bios	preserve state of BIOS ROM. It is default. Driver does not enable BIOS if BIOS was not enabled before.

continues on next page

Table 1 – continued from previous page

noinit	tells driver, that devices were already initialized. You should use it if you have G100 and/or if driver cannot detect memory, you see strange pattern on screen and so on. Devices not enabled by BIOS are still initialized. It is default.
init	driver initializes every device it knows about.

continues on next page

Table 1 – continued from previous page

memtype	specifies memory type, implies 'init'. This is valid only for G200 and G400 and has following meaning: **G200:** • 0 -> 2x128Kx32 chips, 2MB onboard, probably sgram • 1 -> 2x128Kx32 chips, 4MB onboard, probably sgram • 2 -> 2x256Kx32 chips, 4MB onboard, probably sgram • 3 -> 2x256Kx32 chips, 8MB onboard, probably sgram • 4 -> 2x512Kx16 chips, 8/16MB onboard, probably sdram only • 5 -> same as above • 6 -> 4x128Kx32 chips, 4MB onboard, probably sgram • 7 -> 4x128Kx32 chips, 8MB onboard, probably sgram **G400:** • 0 -> 2x512Kx16 SDRAM, 16/32MB • 2x512Kx32 SGRAM, 16/32MB • 1 -> 2x256Kx32 SGRAM, 8/16MB • 2 -> 4x128Kx32 SGRAM, 8/16MB • 3 -> 4x512Kx32 SDRAM, 32MB • 4 -> 4x256Kx32 SGRAM, 16/32MB • 5 -> 2x1Mx32 SDRAM, 32MB • 6 -> reserved • 7 -> reserved You should use sdram or sgram parameter in addition to memtype parameter.

continues on next page

Table 1 – continued from previous page

nomtrr	disables write combining on frame buffer. This slows down driver but there is reported minor incompatibility between GUS DMA and XFree under high loads if write combining is enabled (sound dropouts).
mtrr	enables write combining on frame buffer. It speeds up video accesses much. It is default. You must have MTRR support enabled in kernel and your CPU must have MTRR (f.e. Pentium II have them).
sgram	tells to driver that you have Gxx0 with SGRAM memory. It has no effect without *init*.
sdram	tells to driver that you have Gxx0 with SDRAM memory. It is a default.
inv24	change timings parameters for 24bpp modes on Millennium and Millennium II. Specify this if you see strange color shadows around characters.
noinv24	use standard timings. It is the default.
inverse	invert colors on screen (for LCD displays)
noinverse	show true colors on screen. It is default.
dev:X	bind driver to device X. Driver numbers device from 0 up to N, where device 0 is first *known* device found, 1 second and so on. lspci lists devices in this order. Default is *every* known device.
nohwcursor	disables hardware cursor (use software cursor instead).
hwcursor	enables hardware cursor. It is default. If you are using non-accelerated mode (*noaccel* or *fbset -accel false*), software cursor is used (except for text mode).
noblink	disables cursor blinking. Cursor in text mode always blinks (hw limitation).
blink	enables cursor blinking. It is default.
nofastfont	disables fastfont feature. It is default.
fastfont:X	enables fastfont feature. X specifies size of memory reserved for font data, it must be >= (fontwidth*fontheight*chars_in_font)/8. It is faster on Gx00 series, but slower on older cards.
grayscale	enable grayscale summing. It works in PSEUDOCOLOR modes (text, 4bpp, 8bpp). In DIRECTCOLOR modes it is limited to characters displayed through putc/putcs. Direct accesses to framebuffer can paint colors.
nograyscale	disable grayscale summing. It is default.
cross4MB	enables that pixel line can cross 4MB boundary. It is default for non-Millennium.

continues on next page

Table 1 – continued from previous page

nocross4MB	pixel line must not cross 4MB boundary. It is default for Millennium I or II, because of these devices have hardware limitations which do not allow this. But this option is incompatible with some (if not all yet released) versions of XF86_FBDev.
dfp	enables digital flat panel interface. This option is incompatible with secondary (TV) output - if DFP is active, TV output must be inactive and vice versa. DFP always uses same timing as primary (monitor) output.
dfp:X	use settings X for digital flat panel interface. X is number from 0 to 0xFF, and meaning of each individual bit is described in G400 manual, in description of DAC register 0x1F. For normal operation you should set all bits to zero, except lowest bit. This lowest bit selects who is source of display clocks, whether G400, or panel. Default value is now read back from hardware - so you should specify this value only if you are also using *init* parameter.
outputs:XYZ	set mapping between CRTC and outputs. Each letter can have value of 0 (for no CRTC), 1 (CRTC1) or 2 (CRTC2), and first letter corresponds to primary analog output, second letter to the secondary analog output and third letter to the DVI output. Default setting is 100 for cards below G400 or G400 without DFP, 101 for G400 with DFP, and 111 for G450 and G550. You can set mapping only on first card, use matroxset for setting up other devices.
vesa:X	selects startup videomode. X is number from 0 to 0x1FF, see table above for detailed explanation. Default is 640x480x8bpp if driver has 8bpp support. Otherwise first available of 640x350x4bpp, 640x480x15bpp, 640x480x24bpp, 640x480x32bpp or 80x25 text (80x25 text is always available).

If you are not satisfied with videomode selected by *vesa* option, you can modify it with these options:

xres:X	horizontal resolution, in pixels. Default is derived from *vesa* option.
yres:X	vertical resolution, in pixel lines. Default is derived from *vesa* option.
upper:X	top boundary: lines between end of VSYNC pulse and start of first pixel line of picture. Default is derived from *vesa* option.
lower:X	bottom boundary: lines between end of picture and start of VSYNC pulse. Default is derived from *vesa* option.
vslen:X	length of VSYNC pulse, in lines. Default is derived from *vesa* option.
left:X	left boundary: pixels between end of HSYNC pulse and first pixel. Default is derived from *vesa* option.
right:X	right boundary: pixels between end of picture and start of HSYNC pulse. Default is derived from *vesa* option.
hslen:X	length of HSYNC pulse, in pixels. Default is derived from *vesa* option.
pixclock:X	dotclocks, in ps (picoseconds). Default is derived from *vesa* option and from *fh* and *fv* options.
sync:X	sync. pulse - bit 0 inverts HSYNC polarity, bit 1 VSYNC polarity. If bit 3 (value 0x08) is set, composite sync instead of HSYNC is generated. If bit 5 (value 0x20) is set, sync on green is turned on. Do not forget that if you want sync on green, you also probably want composite sync. Default depends on *vesa*.
depth:X	Bits per pixel: 0=text, 4,8,15,16,24 or 32. Default depends on *vesa*.

If you know capabilities of your monitor, you can specify some (or all) of *maxclk*, *fh* and *fv*. In this case, *pixclock* is computed so that pixclock <= maxclk, real_fh <= fh and real_fv <= fv.

maxclk:X	maximum dotclock. X can be specified in MHz, kHz or Hz. Default is *don`t care*.
fh:X	maximum horizontal synchronization frequency. X can be specified in kHz or Hz. Default is *don't care*.
fv:X	maximum vertical frequency. X must be specified in Hz. Default is 70 for modes derived from *vesa* with yres <= 400, 60Hz for yres > 400.

19.5 Limitations

There are known and unknown bugs, features and misfeatures. Currently there are following known bugs:

- SVGALib does not restore screen on exit
- generic fbcon-cfbX procedures do not work on Alphas. Due to this, *noaccel* (and cfb4 accel) driver does not work on Alpha. So everyone with access to */dev/fb** on Alpha can hang machine (you should restrict access to */dev/fb** - everyone with access to this device can destroy your monitor, believe me...).
- 24bpp does not support correctly XF-FBDev on big-endian architectures.
- interlaced text mode is not supported; it looks like hardware limitation, but I'm not sure.
- Gxx0 SGRAM/SDRAM is not autodetected.
- maybe more...

And following misfeatures:

- SVGALib does not restore screen on exit.

- pixclock for text modes is limited by hardware to
  - 83 MHz on G200
  - 66 MHz on Millennium I
  - 60 MHz on Millennium II

  Because I have no access to other devices, I do not know specific frequencies for them. So driver does not check this and allows you to set frequency higher that this. It causes sparks, black holes and other pretty effects on screen. Device was not destroyed during tests. :-)
- my Millennium G200 oscillator has frequency range from 35 MHz to 380 MHz (and it works with 8bpp on about 320 MHz dotclocks (and changed mclk)). But Matrox says on product sheet that VCO limit is 50-250 MHz, so I believe them (maybe that chip overheats, but it has a very big cooler (G100 has none), so it should work).
- special mixed video/graphics videomodes of Mystique and Gx00 - 2G8V16 and G16V16 are not supported
- color keying is not supported
- feature connector of Mystique and Gx00 is set to VGA mode (it is disabled by BIOS)
- DDC (monitor detection) is supported through dualhead driver
- some check for input values are not so strict how it should be (you can specify vslen=4000 and so on).
- maybe more...

And following features:

- 4bpp is available only on Millennium I and Millennium II. It is hardware limitation.
- selection between 1:5:5:5 and 5:6:5 16bpp videomode is done by -rgba option of fbset: ”fbset -depth 16 -rgba 5,5,5” selects 1:5:5:5, anything else selects 5:6:5 mode.
- text mode uses 6 bit VGA palette instead of 8 bit (one of 262144 colors instead of one of 16M colors). It is due to hardware limitation of Millennium I/II and SVGALib compatibility.

19.6 Benchmarks

It is time to redraw whole screen 1000 times in 1024x768, 60Hz. It is time for draw 6144000 characters on screen through /dev/vcsa (for 32bpp it is about 3GB of data (exactly 3000 MB); for 8x16 font in 16 seconds, i.e. 187 MBps). Times were obtained from one older version of driver, now they are about 3% faster, it is kernel-space only time on P-II/350 MHz, Millennium I in 33 MHz PCI slot, G200 in AGP 2x slot. I did not test vgacon:

```
NOACCEL
      8x16                  12x22
      Millennium I  G200    Millennium I  G200
8bpp    16.42         9.54    12.33         9.13
16bpp   21.00        15.70    19.11        15.02
24bpp   36.66        36.66    35.00        35.00
32bpp   35.00        30.00    33.85        28.66
```

```
ACCEL, nofastfont
      8x16                  12x22                 6x11
      Millennium I  G200   Millennium I  G200   Millennium I  G200
8bpp     7.79         7.24   13.55         7.78   30.00        21.01
16bpp    9.13         7.78   16.16         7.78   30.00        21.01
24bpp   14.17        10.72   18.69        10.24   34.99        21.01
32bpp   16.15      16.16   18.73        13.09   34.99        21.01

ACCEL, fastfont
      8x16                  12x22                 6x11
      Millennium I  G200   Millennium I  G200   Millennium I  G200
8bpp     8.41         6.01    6.54         4.37   16.00        10.51
16bpp    9.54         9.12    8.76         6.17   17.52        14.01
24bpp   15.00        12.36   11.67        10.00   22.01        18.32
32bpp   16.18        18.29*  12.71        12.74   24.44        21.00

TEXT
      8x16
      Millennium I  G200
TEXT     3.29         1.50

* Yes, it is slower than Millennium I.
```

19.7 Dualhead G400

Driver supports dualhead G400 with some limitations:

- secondary head shares videomemory with primary head. It is not problem if you have 32MB of videoram, but if you have only 16MB, you may have to think twice before choosing videomode (for example twice 1880x1440x32bpp is not possible).
- due to hardware limitation, secondary head can use only 16 and 32bpp videomodes.
- secondary head is not accelerated. There were bad problems with accelerated XFree when secondary head used to use acceleration.
- secondary head always powerups in 640x480@60-32 videomode. You have to use fbset to change this mode.
- secondary head always powerups in monitor mode. You have to use fbmatroxset to change it to TV mode. Also, you must select at least 525 lines for NTSC output and 625 lines for PAL output.
- kernel is not fully multihead ready. So some things are impossible to do.
- if you compiled it as module, you must insert i2c-matroxfb, matroxfb_maven and matroxfb_crtc2 into kernel.

19.8 Dualhead G450

Driver supports dualhead G450 with some limitations:

- secondary head shares videomemory with primary head. It is not problem if you have 32MB of videoram, but if you have only 16MB, you may have to think twice before choosing videomode.
- due to hardware limitation, secondary head can use only 16 and 32bpp videomodes.
- secondary head is not accelerated.
- secondary head always powerups in 640x480@60-32 videomode. You have to use fbset to change this mode.
- TV output is not supported
- kernel is not fully multihead ready, so some things are impossible to do.
- if you compiled it as module, you must insert matroxfb_g450 and matroxfb_crtc2 into kernel.

Petr Vandrovec <vandrove@vc.cvut.cz>

CHAPTER

TWENTY

METRONOMEFB

Maintained by Jaya Kumar <jayakumar.lkml.gmail.com>

Last revised: Mar 10, 2008

Metronomefb is a driver for the Metronome display controller. The controller is from E-Ink Corporation. It is intended to be used to drive the E-Ink Vizplex display media. E-Ink hosts some details of this controller and the display media here http://www.e-ink.com/products/matrix/metronome.html .

Metronome is interfaced to the host CPU through the AMLCD interface. The host CPU generates the control information and the image in a framebuffer which is then delivered to the AMLCD interface by a host specific method. The display and error status are each pulled through individual GPIOs.

Metronomefb is platform independent and depends on a board specific driver to do all physical IO work. Currently, an example is implemented for the PXA board used in the AM-200 EPD devkit. This example is am200epd.c

Metronomefb requires waveform information which is delivered via the AMLCD interface to the metronome controller. The waveform information is expected to be delivered from userspace via the firmware class interface. The waveform file can be compressed as long as your udev or hotplug script is aware of the need to uncompress it before delivering it. metronomefb will ask for metronome.wbf which would typically go into /lib/firmware/metronome.wbf depending on your udev/hotplug setup. I have only tested with a single waveform file which was originally labeled 23P01201_60_WT0107_MTC. I do not know what it stands for. Caution should be exercised when manipulating the waveform as there may be a possibility that it could have some permanent effects on the display media. I neither have access to nor know exactly what the waveform does in terms of the physical media.

Metronomefb uses the deferred IO interface so that it can provide a memory mappable frame buffer. It has been tested with tinyx (Xfbdev). It is known to work at this time with xeyes, xclock, xloadimage, xpdf.

CHAPTER

TWENTYONE

MODEDB DEFAULT VIDEO MODE SUPPORT

Currently all frame buffer device drivers have their own video mode databases, which is a mess and a waste of resources. The main idea of modedb is to have

- one routine to probe for video modes, which can be used by all frame buffer devices
- one generic video mode database with a fair amount of standard videomodes (taken from XFree86)
- the possibility to supply your own mode database for graphics hardware that needs non-standard modes, like amifb and Mac frame buffer drivers (which use macmodes.c)

When a frame buffer device receives a video= option it doesn't know, it should consider that to be a video mode option. If no frame buffer device is specified in a video= option, fbmem considers that to be a global video mode option.

Valid mode specifiers (mode_option argument):

```
<xres>x<yres>[M][R][-<bpp>][@<refresh>][i][m][eDd]
<name>[-<bpp>][@<refresh>]
```

with <xres>, <yres>, <bpp> and <refresh> decimal numbers and <name> a string. Things between square brackets are optional.

Valid names are:

```
- NSTC: 480i output, with the CCIR System-M TV mode and NTSC color encoding
- NTSC-J: 480i output, with the CCIR System-M TV mode, the NTSC color
  encoding, and a black level equal to the blanking level.
- PAL: 576i output, with the CCIR System-B TV mode and PAL color encoding
- PAL-M: 480i output, with the CCIR System-M TV mode and PAL color encoding
```

If 'M' is specified in the mode_option argument (after <yres> and before <bpp> and <refresh>, if specified) the timings will be calculated using VESA(TM) Coordinated Video Timings instead of looking up the mode from a table. If 'R' is specified, do a 'reduced blanking' calculation for digital displays. If 'i' is specified, calculate for an interlaced mode. And if 'm' is specified, add margins to the calculation (1.8% of xres rounded down to 8 pixels and 1.8% of yres).

Sample usage: 1024x768M@60m - CVT timing with margins

DRM drivers also add options to enable or disable outputs:

'e' will force the display to be enabled, i.e. it will override the detection if a display is connected. 'D' will force the display to be enabled and use digital output. This is useful for outputs that

have both analog and digital signals (e.g. HDMI and DVI-I). For other outputs it behaves like 'e'. If 'd' is specified the output is disabled.

You can additionally specify which output the options matches to. To force the VGA output to be enabled and drive a specific mode say:

```
video=VGA-1:1280x1024@60me
```

Specifying the option multiple times for different ports is possible, e.g.:

```
video=LVDS-1:d video=HDMI-1:D
```

Options can also be passed after the mode, using commas as separator.

Sample usage: 720x480,rotate=180 - 720x480 mode, rotated by 180 degrees

Valid options are:

```
- margin_top, margin_bottom, margin_left, margin_right (integer):
  Number of pixels in the margins, typically to deal with overscan on TVs
- reflect_x (boolean): Perform an axial symmetry on the X axis
- reflect_y (boolean): Perform an axial symmetry on the Y axis
- rotate (integer): Rotate the initial framebuffer by x
  degrees. Valid values are 0, 90, 180 and 270.
- tv_mode: Analog TV mode. One of "NTSC", "NTSC-443", "NTSC-J", "PAL",
  "PAL-M", "PAL-N", or "SECAM".
- panel_orientation, one of "normal", "upside_down", "left_side_up", or
  "right_side_up". For KMS drivers only, this sets the "panel orientation"
  property on the kms connector as hint for kms users.
```

21.1 What is the VESA(TM) Coordinated Video Timings (CVT)?

From the VESA(TM) Website:

> **"The purpose of CVT is to provide a method for generating a consistent** and coordinated set of standard formats, display refresh rates, and timing specifications for computer display products, both those employing CRTs, and those using other display technologies. The intention of CVT is to give both source and display manufacturers a common set of tools to enable new timings to be developed in a consistent manner that ensures greater compatibility."

This is the third standard approved by VESA(TM) concerning video timings. The first was the Discrete Video Timings (DVT) which is a collection of pre-defined modes approved by VESA(TM). The second is the Generalized Timing Formula (GTF) which is an algorithm to calculate the timings, given the pixelclock, the horizontal sync frequency, or the vertical refresh rate.

The GTF is limited by the fact that it is designed mainly for CRT displays. It artificially increases the pixelclock because of its high blanking requirement. This is inappropriate for digital display interface with its high data rate which requires that it conserves the pixelclock as much as possible. Also, GTF does not take into account the aspect ratio of the display.

The CVT addresses these limitations. If used with CRT's, the formula used is a derivation of GTF with a few modifications. If used with digital displays, the ”reduced blanking” calculation can be used.

From the framebuffer subsystem perspective, new formats need not be added to the global mode database whenever a new mode is released by display manufacturers. Specifying for CVT will work for most, if not all, relatively new CRT displays and probably with most flatpanels, if 'reduced blanking' calculation is specified. (The CVT compatibility of the display can be determined from its EDID. The version 1.3 of the EDID has extra 128-byte blocks where additional timing information is placed. As of this time, there is no support yet in the layer to parse this additional blocks.)

CVT also introduced a new naming convention (should be seen from dmesg output):

```
<pix>M<a>[-R]

where: pix = total amount of pixels in MB (xres x yres)
       M   = always present
       a   = aspect ratio (3 - 4:3; 4 - 5:4; 9 - 15:9, 16:9; A - 16:10)
      -R   = reduced blanking

      example:  .48M3-R - 800x600 with reduced blanking
```

Note: VESA(TM) has restrictions on what is a standard CVT timing:

- aspect ratio can only be one of the above values
- acceptable refresh rates are 50, 60, 70 or 85 Hz only
- if reduced blanking, the refresh rate must be at 60Hz

If one of the above are not satisfied, the kernel will print a warning but the timings will still be calculated.

To find a suitable video mode, you just call:

```
int __init fb_find_mode(struct fb_var_screeninfo *var,
                        struct fb_info *info, const char *mode_option,
                        const struct fb_videomode *db, unsigned int dbsize,
                        const struct fb_videomode *default_mode,
                        unsigned int default_bpp)
```

with db/dbsize your non-standard video mode database, or NULL to use the standard video mode database.

fb_find_mode() first tries the specified video mode (or any mode that matches, e.g. there can be multiple 640x480 modes, each of them is tried). If that fails, the default mode is tried. If that fails, it walks over all modes.

To specify a video mode at bootup, use the following boot options:

```
video=<driver>:<xres>x<yres>[-<bpp>][@refresh]
```

where <driver> is a name from the table below. Valid default modes can be found in drivers/video/fbdev/core/modedb.c. Check your driver's documentation. There may be more modes:

```
Drivers that support modedb boot options
Boot Name     Cards Supported

amifb       - Amiga chipset frame buffer
aty128fb    - ATI Rage128 / Pro frame buffer
atyfb       - ATI Mach64 frame buffer
pm2fb       - Permedia 2/2V frame buffer
pm3fb       - Permedia 3 frame buffer
sstfb       - Voodoo 1/2 (SST1) chipset frame buffer
tdfxfb      - 3D Fx frame buffer
tridentfb   - Trident (Cyber)blade chipset frame buffer
vt8623fb    - VIA 8623 frame buffer
```

BTW, only a few fb drivers use this at the moment. Others are to follow (feel free to send patches). The DRM drivers also support this.

CHAPTER

TWENTYTWO

WHAT IS PVR2FB?

This is a driver for PowerVR 2 based graphics frame buffers, such as the one found in the Dreamcast.

Advantages:

- It provides a nice large console (128 cols + 48 lines with 1024x768) without using tiny, unreadable fonts (NOT on the Dreamcast)
- You can run XF86_FBDev on top of /dev/fb0
- Most important: boot logo :-)

Disadvantages:

- Driver is largely untested on non-Dreamcast systems.

22.1 Configuration

You can pass kernel command line options to pvr2fb with *video=pvr2fb:option1,option2:value2,option3* (multiple options should be separated by comma, values are separated from options by *:*).

Accepted options:

font:X	default font to use. All fonts are supported, including the SUN12x22 font which is very nice at high resolutions.
mode:X	default video mode with format [xres]x[yres]-<bpp>@<refresh rate> The following video modes are supported: 640x640-16@60, 640x480-24@60, 640x480-32@60. The Dreamcast defaults to 640x480-16@60. At the time of writing the 24bpp and 32bpp modes function poorly. Work to fix that is ongoing Note: the 640x240 mode is currently broken, and should not be used for any reason. It is only mentioned here as a reference.
inverse	invert colors on screen (for LCD displays)
nomtrr	disables write combining on frame buffer. This slows down driver but there is reported minor incompatibility between GUS DMA and XFree under high loads if write combining is enabled (sound dropouts). MTRR is enabled by default on systems that have it configured and that support it.
cable:X	cable type. This can be any of the following: vga, rgb, and composite. If none is specified, we guess.
output:X	output type. This can be any of the following: pal, ntsc, and vga. If none is specified, we guess.

22.2 X11

XF86_FBDev has been shown to work on the Dreamcast in the past - though not yet on any 2.6 series kernel.

Paul Mundt <lethal@linuxdc.org>

Updated by Adrian McMenamin <adrian@mcmen.demon.co.uk>

CHAPTER

TWENTYTHREE

DRIVER FOR PXA25X LCD CONTROLLER

The driver supports the following options, either via options=<OPTIONS> when modular or video=pxafb:<OPTIONS> when built in.

For example:

```
modprobe pxafb options=vmem:2M,mode:640x480-8,passive
```

or on the kernel command line:

```
video=pxafb:vmem:2M,mode:640x480-8,passive
```

vmem: VIDEO_MEM_SIZE

Amount of video memory to allocate (can be suffixed with K or M for kilobytes or megabytes)

mode:XRESxYRES[-BPP]

XRES == LCCR1_PPL + 1

YRES == LLCR2_LPP + 1

The resolution of the display in pixels

BPP == The bit depth. Valid values are 1, 2, 4, 8 and 16.

pixclock:PIXCLOCK

Pixel clock in picoseconds

left:LEFT == LCCR1_BLW + 1

right:RIGHT == LCCR1_ELW + 1

hsynclen:HSYNC == LCCR1_HSW + 1

upper:UPPER == LCCR2_BFW

lower:LOWER == LCCR2_EFR

vsynclen:VSYNC == LCCR2_VSW + 1

Display margins and sync times

color | mono => LCCR0_CMS

umm...

active | passive => LCCR0_PAS

Active (TFT) or Passive (STN) display

single | dual => LCCR0_SDS

Single or dual panel passive display

4pix | 8pix => LCCR0_DPD

4 or 8 pixel monochrome single panel data

hsync:HSYNC, vsync:VSYNC

Horizontal and vertical sync. 0 => active low, 1 => active high.

dpc:DPC

Double pixel clock. 1=>true, 0=>false

outputen:POLARITY

Output Enable Polarity. 0 => active low, 1 => active high

pixclockpol:POLARITY

pixel clock polarity 0 => falling edge, 1 => rising edge

23.1 Overlay Support for PXA27x and later LCD controllers

PXA27x and later processors support overlay1 and overlay2 on-top of the base framebuffer (although under-neath the base is also possible). They support palette and no-palette RGB formats, as well as YUV formats (only available on overlay2). These overlays have dedicated DMA channels and behave in a similar way as a framebuffer.

However, there are some differences between these overlay framebuffers and normal framebuffers, as listed below:

1. overlay can start at a 32-bit word aligned position within the base framebuffer, which means they have a start (x, y). This information is encoded into var->nonstd (no, var->xoffset and var->yoffset are not for such purpose).
2. overlay framebuffer is allocated dynamically according to specified 'struct fb_var_screeninfo', the amount is decided by:

```
var->xres_virtual * var->yres_virtual * bpp
```

bpp = 16 -- for RGB565 or RGBT555

bpp = 24 -- for YUV444 packed

bpp = 24 -- for YUV444 planar

bpp = 16 -- for YUV422 planar (1 pixel = 1 Y + 1/2 Cb + 1/2 Cr)

bpp = 12 -- for YUV420 planar (1 pixel = 1 Y + 1/4 Cb + 1/4 Cr)

NOTE:

a. overlay does not support panning in x-direction, thus var->xres_virtual will always be equal to var->xres

b. line length of overlay(s) must be on a 32-bit word boundary, for YUV planar modes, it is a requirement for the component with minimum bits per pixel, e.g. for YUV420, Cr component for one pixel is actually 2-bits, it means the line length should be a multiple of 16-pixels

c. starting horizontal position (XPOS) should start on a 32-bit word boundary, otherwise the fb_check_var() will just fail.

d. the rectangle of the overlay should be within the base plane, otherwise fail

Applications should follow the sequence below to operate an overlay framebuffer:

a. open("/dev/fb[1-2]", ...)

b. ioctl(fd, FBIOGET_VSCREENINFO, ...)

c. modify 'var' with desired parameters:

1) var->xres and var->yres

2) larger var->yres_virtual if more memory is required, usually for double-buffering

3) var->nonstd for starting (x, y) and color format

4) var->{red, green, blue, transp} if RGB mode is to be used

d. ioctl(fd, FBIOPUT_VSCREENINFO, ...)

e. ioctl(fd, FBIOGET_FSCREENINFO, ...)

f. mmap

g. ...

3. for YUV planar formats, these are actually not supported within the framebuffer framework, application has to take care of the offsets and lengths of each component within the framebuffer.

4. var->nonstd is used to pass starting (x, y) position and color format, the detailed bit fields are shown below:

```
31                 23  20          10           0
 +-----------------+---+-----------+-----------+
 |  ... unused ... |FOR|   XPOS    |   YPOS    |
 +-----------------+---+-----------+-----------+
```

FOR - color format, as defined by OVERLAY_FORMAT_* in pxafb.h

- 0 - RGB
- 1 - YUV444 PACKED
- 2 - YUV444 PLANAR
- 3 - YUV422 PLANAR
- 4 - YUR420 PLANAR

XPOS - starting horizontal position

YPOS - starting vertical position

CHAPTER

TWENTYFOUR

S3FB - FBDEV DRIVER FOR S3 TRIO/VIRGE CHIPS

24.1 Supported Hardware

S3 Trio32 S3 Trio64 (and variants V+, UV+, V2/DX, V2/GX) S3 Virge (and variants VX, DX, GX and GX2+) S3 Plato/PX (completely untested) S3 Aurora64V+ (completely untested)

- only PCI bus supported
- only BIOS initialized VGA devices supported
- probably not working on big endian

I tested s3fb on Trio64 (plain, V+ and V2/DX) and Virge (plain, VX, DX), all on i386.

24.2 Supported Features

- 4 bpp pseudocolor modes (with 18bit palette, two variants)
- 8 bpp pseudocolor mode (with 18bit palette)
- 16 bpp truecolor modes (RGB 555 and RGB 565)
- 24 bpp truecolor mode (RGB 888) on (only on Virge VX)
- 32 bpp truecolor mode (RGB 888) on (not on Virge VX)
- text mode (activated by bpp = 0)
- interlaced mode variant (not available in text mode)
- doublescan mode variant (not available in text mode)
- panning in both directions
- suspend/resume support
- DPMS support

Text mode is supported even in higher resolutions, but there is limitation to lower pixclocks (maximum usually between 50-60 MHz, depending on specific hardware, i get best results from plain S3 Trio32 card - about 75 MHz). This limitation is not enforced by driver. Text mode supports 8bit wide fonts only (hardware limitation) and 16bit tall fonts (driver limitation). Text mode support is broken on S3 Trio64 V2/DX.

There are two 4 bpp modes. First mode (selected if nonstd == 0) is mode with packed pixels, high nibble first. Second mode (selected if nonstd == 1) is mode with interleaved planes (1 byte interleave), MSB first. Both modes support 8bit wide fonts only (driver limitation).

Suspend/resume works on systems that initialize video card during resume and if device is active (for example used by fbcon).

24.3 Missing Features

(alias TODO list)

- secondary (not initialized by BIOS) device support
- big endian support
- Zorro bus support
- MMIO support
- 24 bpp mode support on more cards
- support for fontwidths != 8 in 4 bpp modes
- support for fontheight != 16 in text mode
- composite and external sync (is anyone able to test this?)
- hardware cursor
- video overlay support
- vsync synchronization
- feature connector support
- acceleration support (8514-like 2D, Virge 3D, busmaster transfers)
- better values for some magic registers (performance issues)

24.4 Known bugs

- cursor disable in text mode doesn't work
- text mode broken on S3 Trio64 V2/DX

-- Ondrej Zajicek <santiago@crfreenet.org>

CHAPTER

TWENTYFIVE

WHAT IS SA1100FB?

This is a driver for a graphic framebuffer for the SA-1100 LCD controller.

25.1 Configuration

For most common passive displays, giving the option:

```
video=sa1100fb:bpp:<value>,lccr0:<value>,lccr1:<value>,lccr2:<value>,lccr3:
↪<value>
```

on the kernel command line should be enough to configure the controller. The bits per pixel (bpp) value should be 4, 8, 12, or 16. LCCR values are display-specific and should be computed as documented in the SA-1100 Developer's Manual, Section 11.7. Dual-panel displays are supported as long as the SDS bit is set in LCCR0; GPIO<9:2> are used for the lower panel.

For active displays or displays requiring additional configuration (controlling backlights, powering on the LCD, etc.), the command line options may not be enough to configure the display. Adding sections to sa1100fb_init_fbinfo(), sa1100fb_activate_var(), sa1100fb_disable_lcd_controller(), and sa1100fb_enable_lcd_controller() will probably be necessary.

Accepted options:

```
bpp:<value>     Configure for <value> bits per pixel
lccr0:<value>   Configure LCD control register 0 (11.7.3)
lccr1:<value>   Configure LCD control register 1 (11.7.4)
lccr2:<value>   Configure LCD control register 2 (11.7.5)
lccr3:<value>   Configure LCD control register 3 (11.7.6)
```

Mark Huang <mhuang@livetoy.com>

CHAPTER

TWENTYSIX

SH7760/SH7763 INTEGRATED LCDC FRAMEBUFFER DRIVER

26.1 0. Overview

The SH7760/SH7763 have an integrated LCD Display controller (LCDC) which supports (in theory) resolutions ranging from 1x1 to 1024x1024, with color depths ranging from 1 to 16 bits, on STN, DSTN and TFT Panels.

Caveats:

- Framebuffer memory must be a large chunk allocated at the top of Area3 (HW requirement). Because of this requirement you should NOT make the driver a module since at runtime it may become impossible to get a large enough contiguous chunk of memory.
- The driver does not support changing resolution while loaded (displays aren't hotpluggable anyway)
- Heavy flickering may be observed a) if you're using 15/16bit color modes at >= 640x480 px resolutions, b) during PCMCIA (or any other slow bus) activity.
- Rotation works only 90degress clockwise, and only if horizontal resolution is <= 320 pixels.

Files:

- drivers/video/sh7760fb.c
- include/asm-sh/sh7760fb.h
- *SH7760/SH7763 integrated LCDC Framebuffer driver*

26.2 1. Platform setup

SH7760:
Video data is fetched via the DMABRG DMA engine, so you have to configure the SH DMAC for DMABRG mode (write 0x94808080 to the DMARSRA register somewhere at boot).

PFC registers PCCR and PCDR must be set to peripheral mode. (write zeros to both).

The driver does NOT do the above for you since board setup is, well, job of the board setup code.

26.3 2. Panel definitions

The LCDC must explicitly be told about the type of LCD panel attached. Data must be wrapped in a ”struct sh7760fb_platdata” and passed to the driver as platform_data.

Suggest you take a closer look at the SH7760 Manual, Section 30. (http://documentation.renesas.com/eng/products/mpumcu/e602291_sh7760.pdf)

The following code illustrates what needs to be done to get the framebuffer working on a 640x480 TFT:

```
#include <linux/fb.h>
#include <asm/sh7760fb.h>

/*
 * NEC NL6440bc26-01 640x480 TFT
 * dotclock 25175 kHz
 * Xres                 640     Yres            480
 * Htotal      800      Vtotal          525
 * HsynStart   656      VsynStart       490
 * HsynLenn    30       VsynLenn        2
 *
 * The linux framebuffer layer does not use the syncstart/synclen
 * values but right/left/upper/lower margin values. The comments
 * for the x_margin explain how to calculate those from given
 * panel sync timings.
 */
static struct fb_videomode nl6448bc26 = {
       .name           = "NL6448BC26",
       .refresh        = 60,
       .xres           = 640,
       .yres           = 480,
       .pixclock       = 39683,        /* in picoseconds! */
       .hsync_len      = 30,
       .vsync_len      = 2,
       .left_margin    = 114,  /* HTOT - (HSYNSLEN + HSYNSTART) */
       .right_margin   = 16,   /* HSYNSTART - XRES */
       .upper_margin   = 33,   /* VTOT - (VSYNLEN + VSYNSTART) */
       .lower_margin   = 10,   /* VSYNSTART - YRES */
       .sync           = FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT,
       .vmode          = FB_VMODE_NONINTERLACED,
       .flag           = 0,
};

static struct sh7760fb_platdata sh7760fb_nl6448 = {
       .def_mode       = &nl6448bc26,
       .ldmtr          = LDMTR_TFT_COLOR_16,   /* 16bit TFT panel */
       .lddfr          = LDDFR_8BPP,           /* we want 8bit output */
       .ldpmmr         = 0x0070,
       .ldpspr         = 0x0500,
       .ldaclnr        = 0,
       .ldickr         = LDICKR_CLKSRC(LCDC_CLKSRC_EXTERNAL) |
```

```
                        LDICKR_CLKDIV(1),
        .rotate         = 0,
        .novsync        = 1,
        .blank          = NULL,
};

/* SH7760:
 * 0xFE300800: 256 * 4byte xRGB palette ram
 * 0xFE300C00: 42 bytes ctrl registers
 */
static struct resource sh7760_lcdc_res[] = {
        [0] = {
                .start  = 0xFE300800,
                .end    = 0xFE300CFF,
                .flags  = IORESOURCE_MEM,
        },
        [1] = {
                .start  = 65,
                .end    = 65,
                .flags  = IORESOURCE_IRQ,
        },
};

static struct platform_device sh7760_lcdc_dev = {
        .dev    = {
                .platform_data = &sh7760fb_nl6448,
        },
        .name           = "sh7760-lcdc",
        .id             = -1,
        .resource       = sh7760_lcdc_res,
        .num_resources  = ARRAY_SIZE(sh7760_lcdc_res),
};
```

CHAPTER

TWENTYSEVEN

WHAT IS SISFB?

sisfb is a framebuffer device driver for SiS (Silicon Integrated Systems) graphics chips. Supported are:

- SiS 300 series: SiS 300/305, 540, 630(S), 730(S)
- SiS 315 series: SiS 315/H/PRO, 55x, (M)65x, 740, (M)661(F/M)X, (M)741(GX)
- SiS 330 series: SiS 330 ("Xabre"), (M)760

27.1 Why do I need a framebuffer driver?

sisfb is eg. useful if you want a high-resolution text console. Besides that, sisfb is required to run DirectFB (which comes with an additional, dedicated driver for the 315 series).

On the 300 series, sisfb on kernels older than 2.6.3 furthermore plays an important role in connection with DRM/DRI: Sisfb manages the memory heap used by DRM/DRI for 3D texture and other data. This memory management is required for using DRI/DRM.

Kernels >= around 2.6.3 do not need sisfb any longer for DRI/DRM memory management. The SiS DRM driver has been updated and features a memory manager of its own (which will be used if sisfb is not compiled). So unless you want a graphical console, you don't need sisfb on kernels >=2.6.3.

Sidenote: Since this seems to be a commonly made mistake: sisfb and vesafb cannot be active at the same time! Do only select one of them in your kernel configuration.

27.2 How are parameters passed to sisfb?

Well, it depends: If compiled statically into the kernel, use lilo's append statement to add the parameters to the kernel command line. Please see lilo's (or GRUB's) documentation for more information. If sisfb is a kernel module, parameters are given with the modprobe (or insmod) command.

Example for sisfb as part of the static kernel: Add the following line to your lilo.conf:

```
append="video=sisfb:mode:1024x768x16,mem:12288,rate:75"
```

Example for sisfb as a module: Start sisfb by typing:

```
modprobe sisfb mode=1024x768x16 rate=75 mem=12288
```

A common mistake is that folks use a wrong parameter format when using the driver compiled into the kernel. Please note: If compiled into the kernel, the parameter format is video=sisfb:mode:none or video=sisfb:mode:1024x768x16 (or whatever mode you want to use, alternatively using any other format described above or the vesa keyword instead of mode). If compiled as a module, the parameter format reads mode=none or mode=1024x768x16 (or whatever mode you want to use). Using a ”=” for a ”:” (and vice versa) is a huge difference! Additionally: If you give more than one argument to the in-kernel sisfb, the arguments are separated with ”,”. For example:

```
video=sisfb:mode:1024x768x16,rate:75,mem:12288
```

27.3 How do I use it?

Preface statement: This file only covers very little of the driver's capabilities and features. Please refer to the author's and maintainer's website at http://www.winischhofer.net/linuxsisvga.shtml for more information. Additionally, ”modinfo sisfb” gives an overview over all supported options including some explanation.

The desired display mode can be specified using the keyword ”mode” with a parameter in one of the following formats:

- XxYxDepth or
- XxY-Depth or
- XxY-Depth@Rate or
- XxY
- or simply use the VESA mode number in hexadecimal or decimal.

For example: 1024x768x16, 1024x768-16@75, 1280x1024-16. If no depth is specified, it defaults to 8. If no rate is given, it defaults to 60Hz. Depth 32 means 24bit color depth (but 32 bit framebuffer depth, which is not relevant to the user).

Additionally, sisfb understands the keyword ”vesa” followed by a VESA mode number in decimal or hexadecimal. For example: vesa=791 or vesa=0x117. Please use either ”mode” or ”vesa” but not both.

Linux 2.4 only: If no mode is given, sisfb defaults to ”no mode” (mode=none) if compiled as a module; if sisfb is statically compiled into the kernel, it defaults to 800x600x8 unless CRT2 type is LCD, in which case the LCD's native resolution is used. If you want to switch to a different mode, use the fbset shell command.

Linux 2.6 only: If no mode is given, sisfb defaults to 800x600x8 unless CRT2 type is LCD, in which case it defaults to the LCD's native resolution. If you want to switch to another mode, use the stty shell command.

You should compile in both vgacon (to boot if you remove you SiS card from your system) and sisfb (for graphics mode). Under Linux 2.6, also ”Framebuffer console support” (fbcon) is needed for a graphical console.

You should *not* compile-in vesafb. And please do not use the ”vga=” keyword in lilo's or grub's configuration file; mode selection is done using the ”mode” or ”vesa” keywords as a parameter. See above and below.

27.4 X11

If using XFree86 or X.org, it is recommended that you don't use the ”fbdev” driver but the dedicated ”sis” X driver. The ”sis” X driver and sisfb are developed by the same person (Thomas Winischhofer) and cooperate well with each other.

27.5 SVGALib

SVGALib, if directly accessing the hardware, never restores the screen correctly, especially on laptops or if the output devices are LCD or TV. Therefore, use the chipset ”FBDEV” in SVGALib configuration. This will make SVGALib use the framebuffer device for mode switches and restoration.

27.6 Configuration

(Some) accepted options:

off	Disable sisfb. This option is only understood if sisfb is in-kernel, not a module.
mem:X	size of memory for the console, rest will be used for DRI/DRM. X is in kilobytes. On 300 series, the default is 4096, 8192 or 16384 (each in kilobyte) depending on how much video ram the card has. On 315/330 series, the default is the maximum available ram (since DRI/DRM is not supported for these chipsets).
noaccel	do not use 2D acceleration engine. (Default: use acceleration)
noypan	disable y-panning and scroll by redrawing the entire screen. This is much slower than y-panning. (Default: use y-panning)
vesa:X	selects startup videomode. X is number from 0 to 0x1FF and represents the VESA mode number (can be given in decimal or hexadecimal form, the latter prefixed with ”0x”).
mode:X	selects startup videomode. Please see above for the format of ”X”.

Boolean options such as ”noaccel” or ”noypan” are to be given without a parameter if sisfb is in-kernel (for example ”video=sisfb:noypan). If sisfb is a module, these are to be set to 1 (for example ”modprobe sisfb noypan=1”).

Thomas Winischhofer <thomas@winischhofer.net>

May 27, 2004

CHAPTER

TWENTYEIGHT

SM501FB

Configuration:

You can pass the following kernel command line options to sm501 videoframebuffer:

```
sm501fb.bpp=    SM501 Display driver:
                Specify bits-per-pixel if not specified by 'mode'

sm501fb.mode=   SM501 Display driver:
                Specify resolution as
                "<xres>x<yres>[-<bpp>][@<refresh>]"
```

CHAPTER

TWENTYNINE

WHAT IS SM712FB?

This is a graphics framebuffer driver for Silicon Motion SM712 based processors.

29.1 How to use it?

Switching modes is done using the video=sm712fb:... boot parameter.

If you want, for example, enable a resolution of 1280x1024x24bpp you should pass to the kernel this command line: ”video=sm712fb:0x31B”.

You should not compile-in vesafb.

Currently supported video modes are:

29.1.1 Graphic modes

bpp	640x480	800x600	1024x768	1280x1024
8	0x301	0x303	0x305	0x307
16	0x311	0x314	0x317	0x31A
24	0x312	0x315	0x318	0x31B

29.2 Missing Features

(alias TODO list)

- 2D acceleration
- dual-head support

CHAPTER

THIRTY

SSTFB

30.1 Introduction

This is a frame buffer device driver for 3dfx' Voodoo Graphics (aka voodoo 1, aka sst1) and Voodoo^2 (aka Voodoo 2, aka CVG) based video boards. It's highly experimental code, but is guaranteed to work on my computer, with my ”Maxi Gamer 3D” and ”Maxi Gamer 3d^2” boards, and with me ”between chair and keyboard”. Some people tested other combinations and it seems that it works. The main page is located at <http://sstfb.sourceforge.net>, and if you want the latest version, check out the CVS, as the driver is a work in progress, I feel uncomfortable with releasing tarballs of something not completely working...Don't worry, it's still more than usable (I eat my own dog food)

Please read the Bug section, and report any success or failure to me (Ghozlane Toumi <gtoumi@laposte.net>). BTW, If you have only one monitor , and you don't feel like playing with the vga passthrou cable, I can only suggest borrowing a screen somewhere...

30.2 Installation

This driver (should) work on ix86, with ”late” 2.2.x kernel (tested with x = 19) and ”recent” 2.4.x kernel, as a module or compiled in. It has been included in mainstream kernel since the infamous 2.4.10. You can apply the patches found in *sstfb/kernel/*-2.{2|4}.x.patch*, and copy sstfb.c to linux/drivers/video/, or apply a single patch, *sstfb/patch-2.{2|4}.x-sstfb-yymmdd* to your linux source tree.

Then configure your kernel as usual: choose ”m” or ”y” to 3Dfx Voodoo Graphics in section ”console”. Compile, install, have fun... and please drop me a report :)

30.3 Module Usage

Warning:

1. You should read completely this section before issuing any command.
2. If you have only one monitor to play with, once you insmod the module, the 3dfx takes control of the output, so you'll have to plug the monitor to the ”normal” video board in

order to issue the commands, or you can blindly use sst_dbg_vgapass in the tools directory (See Tools). The latest solution is pass the parameter vgapass=1 when insmodding the driver. (See Kernel/Modules Options)

30.3.1 Module insertion

1. insmod sstfb.o

   you should see some strange output from the board: a big blue square, a green and a red small squares and a vertical white rectangle. why? the function's name is self-explanatory: ”sstfb_test()”... (if you don't have a second monitor, you'll have to plug your monitor directly to the 2D videocard to see what you're typing)

2. con2fb /dev/fbx /dev/ttyx

   bind a tty to the new frame buffer. if you already have a frame buffer driver, the voodoo fb will likely be /dev/fb1. if not, the device will be /dev/fb0. You can check this by doing a cat /proc/fb. You can find a copy of con2fb in tools/ directory. if you don't have another fb device, this step is superfluous, as the console subsystem automagically binds ttys to the fb.

3. switch to the virtual console you just mapped. ”tadaaa” ...

30.3.2 Module removal

1. con2fb /dev/fbx /dev/ttyx

   bind the tty to the old frame buffer so the module can be removed. (how does it work with vgacon ? short answer : it doesn't work)

2. rmmod sstfb

30.3.3 Kernel/Modules Options

You can pass some options to the sstfb module, and via the kernel command line when the driver is compiled in: for module : insmod sstfb.o option1=value1 option2=value2 ... in kernel : video=sstfb:option1,option2:value2,option3 ...

sstfb supports the following options:

Module	Kernel	Description
vgapass=0	vganopass	Enable or disable VGA passthrou cable.
vgapass=1	vgapass	When enabled, the monitor will get the signal from the VGA board and not from the voodoo. Default: nopass
mem=x	mem:x	Force frame buffer memory in MiB allowed values: 0, 1, 2, 4. Default: 0 (= autodetect)
inverse=1	inverse	Supposed to enable inverse console. doesn't work yet...
clipping=1	clipping	Enable or disable clipping.
clipping=0	noclipping	With clipping enabled, all off-screen reads and writes are discarded. Default: enable clipping.
gfxclk=x	gfxclk:x	Force graphic clock frequency (in MHz). Be careful with this option, it may be DANGEROUS. Default: auto • 50Mhz for Voodoo 1, • 75MHz for Voodoo 2.
slowpci=1	fastpci	Enable or disable fast PCI read/writes.
slowpci=1	slowpci	Default : fastpci
dev=x	dev:x	Attach the driver to device number x. 0 is the first compatible board (in lspci order)

30.4 Tools

These tools are mostly for debugging purposes, but you can find some of these interesting:

- *con2fb*, maps a tty to a fbramebuffer:

```
con2fb /dev/fb1 /dev/tty5
```

- *sst_dbg_vgapass*, changes vga passthrou. You have to recompile the driver with SST_DEBUG and SST_DEBUG_IOCTL set to 1:

```
sst_dbg_vgapass /dev/fb1 1 (enables vga cable)
sst_dbg_vgapass /dev/fb1 0 (disables vga cable)
```

- *glide_reset*, resets the voodoo using glide use this after rmmoding sstfb, if the module refuses to reinsert.

30.5 Bugs

- DO NOT use glide while the sstfb module is in, you'll most likely hang your computer.
- If you see some artefacts (pixels not cleaning and stuff like that), try turning off clipping (clipping=0), and/or using slowpci
- the driver don't detect the 4Mb frame buffer voodoos, it seems that the 2 last Mbs wrap around. looking into that .
- The driver is 16 bpp only, 24/32 won't work.
- The driver is not your_favorite_toy-safe. this includes SMP...

  [Actually from inspection it seems to be safe - Alan]
- When using XFree86 FBdev (X over fbdev) you may see strange color patterns at the border of your windows (the pixels lose the lowest byte -> basically the blue component and some of the green). I'm unable to reproduce this with XFree86-3.3, but one of the testers has this problem with XFree86-4. Apparently recent Xfree86-4.x solve this problem.
- I didn't really test changing the palette, so you may find some weird things when playing with that.
- Sometimes the driver will not recognise the DAC, and the initialisation will fail. This is specifically true for voodoo 2 boards, but it should be solved in recent versions. Please contact me.
- The 24/32 is not likely to work anytime soon, knowing that the hardware does ... unusual things in 24/32 bpp.

30.6 Todo

- Get rid of the previous paragraph.
- Buy more coffee.
- test/port to other arch.
- try to add panning using tweeks with front and back buffer .
- try to implement accel on voodoo2, this board can actually do a lot in 2D even if it was sold as a 3D only board ...

Ghozlane Toumi <gtoumi@laposte.net>

Date: 2002/05/09 20:11:45

http://sstfb.sourceforge.net/README

CHAPTER

THIRTYONE

WHAT IS TGAFB?

This is a driver for DECChip 21030 based graphics framebuffers, a.k.a. TGA cards, which are usually found in older Digital Alpha systems. The following models are supported:

- ZLxP-E1 (8bpp, 2 MB VRAM)
- ZLxP-E2 (32bpp, 8 MB VRAM)
- ZLxP-E3 (32bpp, 16 MB VRAM, Zbuffer)

This version is an almost complete rewrite of the code written by Geert Uytterhoeven, which was based on the original TGA console code written by Jay Estabrook.

Major new features since Linux 2.0.x:

- Support for multiple resolutions
- Support for fixed-frequency and other oddball monitors (by allowing the video mode to be set at boot time)

User-visible changes since Linux 2.2.x:

- Sync-on-green is now handled properly
- More useful information is printed on bootup (this helps if people run into problems)

This driver does not (yet) support the TGA2 family of framebuffers, so the PowerStorm 3D30/4D20 (also known as PBXGB) cards are not supported. These can however be used with the standard VGA Text Console driver.

31.1 Configuration

You can pass kernel command line options to tgafb with *video=tgafb:option1,option2:value2,option3* (multiple options should be separated by comma, values are separated from options by *:*).

Accepted options:

font:X	default font to use. All fonts are supported, including the SUN12x22 font which is very nice at high resolutions.
mode:X	default video mode. The following video modes are supported: 640x480-60, 800x600-56, 640x480-72, 800x600-60, 800x600-72, 1024x768-60, 1152x864-60, 1024x768-70, 1024x768-76, 1152x864-70, 1280x1024-61, 1024x768-85, 1280x1024-70, 1152x864-84, 1280x1024-76, 1280x1024-85

31.2 Known Issues

The XFree86 FBDev server has been reported not to work, since tgafb doesn't do mmap(). Running the standard XF86_TGA server from XFree86 3.3.x works fine for me, however this server does not do acceleration, which make certain operations quite slow. Support for acceleration is being progressively integrated in XFree86 4.x.

When running tgafb in resolutions higher than 640x480, on switching VCs from tgafb to XF86_TGA 3.3.x, the entire screen is not re-drawn and must be manually refreshed. This is an X server problem, not a tgafb problem, and is fixed in XFree86 4.0.

Enjoy!

Martin Lucina <mato@kotelna.sk>

CHAPTER

THIRTYTWO

TRIDENTFB

Tridentfb is a framebuffer driver for some Trident chip based cards.

The following list of chips is thought to be supported although not all are tested:

those from the TGUI series 9440/96XX and with Cyber in their names those from the Image series and with Cyber in their names those with Blade in their names (Blade3D,CyberBlade...) the newer CyberBladeXP family

All families are accelerated. Only PCI/AGP based cards are supported, none of the older Tridents. The driver supports 8, 16 and 32 bits per pixel depths. The TGUI family requires a line length to be power of 2 if acceleration is enabled. This means that range of possible resolutions and bpp is limited comparing to the range if acceleration is disabled (see list of parameters below).

Known bugs:

1. The driver randomly locks up on 3DImage975 chip with acceleration enabled. The same happens in X11 (Xorg).
2. The ramdac speeds require some more fine tuning. It is possible to switch resolution which the chip does not support at some depths for older chips.

32.1 How to use it?

When booting you can pass the video parameter:

```
video=tridentfb
```

The parameters for tridentfb are concatenated with a ':' as in this example:

```
video=tridentfb:800x600-16@75,noaccel
```

The second level parameters that tridentfb understands are:

noaccel	turns off acceleration (when it doesn't work for your card)
fp	use flat panel related stuff
crt	assume monitor is present instead of fp
center	for flat panels and resolutions smaller than native size center the image, otherwise use
stretch	
memsize	integer value in KB, use if your card's memory size is misdetected. look at the driver output to see what it says when initializing.
memdiff	integer value in KB, should be nonzero if your card reports more memory than it actually has. For instance mine is 192K less than detection says in all three BIOS selectable situations 2M, 4M, 8M. Only use if your video memory is taken from main memory hence of configurable size. Otherwise use memsize. If in some modes which barely fit the memory you see garbage at the bottom this might help by not letting change to that mode anymore.
nativex	the width in pixels of the flat panel.If you know it (usually 1024 800 or 1280) and it is not what the driver seems to detect use it.
bpp	bits per pixel (8,16 or 32)
mode	a mode name like 800x600-8@75 as described in *modedb default video mode support*

Using insane values for the above parameters will probably result in driver misbehaviour so take care(for instance memsize=12345678 or memdiff=23784 or nativex=93)

Contact: jani@astechnix.ro

CHAPTER

THIRTYTHREE

WHAT IS UDLFB?

This is a driver for DisplayLink USB 2.0 era graphics chips.

DisplayLink chips provide simple hline/blit operations with some compression, pairing that with a hardware framebuffer (16MB) on the other end of the USB wire. That hardware framebuffer is able to drive the VGA, DVI, or HDMI monitor with no CPU involvement until a pixel has to change.

The CPU or other local resource does all the rendering; optionally compares the result with a local shadow of the remote hardware framebuffer to identify the minimal set of pixels that have changed; and compresses and sends those pixels line-by-line via USB bulk transfers.

Because of the efficiency of bulk transfers and a protocol on top that does not require any acks - the effect is very low latency that can support surprisingly high resolutions with good performance for non-gaming and non-video applications.

Mode setting, EDID read, etc are other bulk or control transfers. Mode setting is very flexible - able to set nearly arbitrary modes from any timing.

Advantages of USB graphics in general:

- Ability to add a nearly arbitrary number of displays to any USB 2.0 capable system. On Linux, number of displays is limited by fbdev interface (FB_MAX is currently 32). Of course, all USB devices on the same host controller share the same 480Mbs USB 2.0 interface.

Advantages of supporting DisplayLink chips with kernel framebuffer interface:

- The actual hardware functionality of DisplayLink chips matches nearly one-to-one with the fbdev interface, making the driver quite small and tight relative to the functionality it provides.
- X servers and other applications can use the standard fbdev interface from user mode to talk to the device, without needing to know anything about USB or DisplayLink's protocol at all. A ”displaylink” X driver and a slightly modified ”fbdev” X driver are among those that already do.

Disadvantages:

- Fbdev's mmap interface assumes a real hardware framebuffer is mapped. In the case of USB graphics, it is just an allocated (virtual) buffer. Writes need to be detected and encoded into USB bulk transfers by the CPU. Accurate damage/changed area notifications work around this problem. In the future, hopefully fbdev will be enhanced with an small standard interface to allow mmap clients to report damage, for the benefit of virtual or remote framebuffers.
- Fbdev does not arbitrate client ownership of the framebuffer well.

- Fbcon assumes the first framebuffer it finds should be consumed for console.
- It's not clear what the future of fbdev is, given the rise of KMS/DRM.

33.1 How to use it?

Udlfb, when loaded as a module, will match against all USB 2.0 generation DisplayLink chips (Alex and Ollie family). It will then attempt to read the EDID of the monitor, and set the best common mode between the DisplayLink device and the monitor's capabilities.

If the DisplayLink device is successful, it will paint a ”green screen” which means that from a hardware and fbdev software perspective, everything is good.

At that point, a /dev/fb? interface will be present for user-mode applications to open and begin writing to the framebuffer of the DisplayLink device using standard fbdev calls. Note that if mmap() is used, by default the user mode application must send down damage notifications to trigger repaints of the changed regions. Alternatively, udlfb can be recompiled with experimental defio support enabled, to support a page-fault based detection mechanism that can work without explicit notification.

The most common client of udlfb is xf86-video-displaylink or a modified xf86-video-fbdev X server. These servers have no real DisplayLink specific code. They write to the standard framebuffer interface and rely on udlfb to do its thing. The one extra feature they have is the ability to report rectangles from the X DAMAGE protocol extension down to udlfb via udlfb's damage interface (which will hopefully be standardized for all virtual framebuffers that need damage info). These damage notifications allow udlfb to efficiently process the changed pixels.

33.2 Module Options

Special configuration for udlfb is usually unnecessary. There are a few options, however.

From the command line, pass options to modprobe:

```
modprobe udlfb fb_defio=0 console=1 shadow=1
```

Or change options on the fly by editing /sys/module/udlfb/parameters/PARAMETER_NAME

```
cd /sys/module/udlfb/parameters
ls # to see a list of parameter names
sudo nano PARAMETER_NAME
# change the parameter in place, and save the file.
```

Unplug/replug USB device to apply with new settings.

Or to apply options permanently, create a modprobe configuration file like /etc/modprobe.d/udlfb.conf with text:

```
options udlfb fb_defio=0 console=1 shadow=1
```

Accepted boolean options:

fb_defio	Make use of the fb_defio (CONFIG_FB_DEFERRED_IO) kernel module to track changed areas of the framebuffer by page faults. Standard fbdev applications that use mmap but that do not report damage, should be able to work with this enabled. Disable when running with X server that supports reporting changed regions via ioctl, as this method is simpler, more stable, and higher performance. default: fb_defio=1
console	Allow fbcon to attach to udlfb provided framebuffers. Can be disabled if fbcon and other clients (e.g. X with --shared-vt) are in conflict. default: console=1
shadow	Allocate a 2nd framebuffer to shadow what's currently across the USB bus in device memory. If any pixels are unchanged, do not transmit. Spends host memory to save USB transfers. Enabled by default. Only disable on very low memory systems. default: shadow=1

33.3 Sysfs Attributes

Udlfb creates several files in /sys/class/graphics/fb? Where ? is the sequential framebuffer id of the particular DisplayLink device

edid	If a valid EDID blob is written to this file (typically by a udev rule), then udlfb will use this EDID as a backup in case reading the actual EDID of the monitor attached to the DisplayLink device fails. This is especially useful for fixed panels, etc. that cannot communicate their capabilities via EDID. Reading this file returns the current EDID of the attached monitor (or last backup value written). This is useful to get the EDID of the attached monitor, which can be passed to utilities like parse-edid.
metrics_bytes_rendered	32-bit count of pixel bytes rendered
metrics_bytes_identical	32-bit count of how many of those bytes were found to be unchanged, based on a shadow framebuffer check
metrics_bytes_sent	32-bit count of how many bytes were transferred over USB to communicate the resulting changed pixels to the hardware. Includes compression and protocol overhead
metrics_cpu_kcycles_used	32-bit count of CPU cycles used in processing the above pixels (in thousands of cycles).
metrics_reset	Write-only. Any write to this file resets all metrics above to zero. Note that the 32-bit counters above roll over very quickly. To get reliable results, design performance tests to start and finish in a very short period of time (one minute or less is safe).

Bernie Thompson <bernie@plugable.com>

CHAPTER

THIRTYFOUR

UVESAFB - A GENERIC DRIVER FOR VBE2+ COMPLIANT VIDEO CARDS

34.1 1. Requirements

uvesafb should work with any video card that has a Video BIOS compliant with the VBE 2.0 standard.

Unlike other drivers, uvesafb makes use of a userspace helper called v86d. v86d is used to run the x86 Video BIOS code in a simulated and controlled environment. This allows uvesafb to function on arches other than x86. Check the v86d documentation for a list of currently supported arches.

v86d source code can be downloaded from the following website:

https://github.com/mjanusz/v86d

Please refer to the v86d documentation for detailed configuration and installation instructions.

Note that the v86d userspace helper has to be available at all times in order for uvesafb to work properly. If you want to use uvesafb during early boot, you will have to include v86d into an initramfs image, and either compile it into the kernel or use it as an initrd.

34.2 2. Caveats and limitations

uvesafb is a _generic_ driver which supports a wide variety of video cards, but which is ultimately limited by the Video BIOS interface. The most important limitations are:

- Lack of any type of acceleration.
- A strict and limited set of supported video modes. Often the native or most optimal resolution/refresh rate for your setup will not work with uvesafb, simply because the Video BIOS doesn't support the video mode you want to use. This can be especially painful with widescreen panels, where native video modes don't have the 4:3 aspect ratio, which is what most BIOS-es are limited to.
- Adjusting the refresh rate is only possible with a VBE 3.0 compliant Video BIOS. Note that many nVidia Video BIOS-es claim to be VBE 3.0 compliant, while they simply ignore any refresh rate settings.

34.3 3. Configuration

uvesafb can be compiled either as a module, or directly into the kernel. In both cases it supports the same set of configuration options, which are either given on the kernel command line or as module parameters, e.g.:

```
video=uvesafb:1024x768-32,mtrr:3,ywrap (compiled into the kernel)

# modprobe uvesafb mode_option=1024x768-32 mtrr=3 scroll=ywrap  (module)
```

Accepted options:

ypan	Enable display panning using the VESA protected mode interface. The visible screen is just a window of the video memory, console scrolling is done by changing the start of the window. This option is available on x86 only and is the default option on that architecture.
ywrap	Same as ypan, but assumes your gfx board can wrap-around the video memory (i.e. starts reading from top if it reaches the end of video memory). Faster than ypan. Available on x86 only.
redraw	Scroll by redrawing the affected part of the screen, this is the default on non-x86.

(If you're using uvesafb as a module, the above three options are used a parameter of the scroll option, e.g. scroll=ypan.)

vgapal	Use the standard VGA registers for palette changes.
pmipal	Use the protected mode interface for palette changes. This is the default if the protected mode interface is available. Available on x86 only.
mtrr:n	Setup memory type range registers for the framebuffer where n: • 0 - disabled (equivalent to nomtrr) • 3 - write-combining (default) Values other than 0 and 3 will result in a warning and will be treated just like 3.
nomtrr	Do not use memory type range registers.
vremap:n	Remap 'n' MiB of video RAM. If 0 or not specified, remap memory according to video mode.
vtotal:n	If the video BIOS of your card incorrectly determines the total amount of video RAM, use this option to override the BIOS (in MiB).
<mode>	The mode you want to set, in the standard modedb format. Refer to *modedb default video mode support* for a detailed description. When uvesafb is compiled as a module, the mode string should be provided as a value of the 'mode_option' option.
vbemode:x	Force the use of VBE mode x. The mode will only be set if it's found in the VBE-provided list of supported modes. NOTE: The mode number 'x' should be specified in VESA mode number notation, not the Linux kernel one (eg. 257 instead of 769). HINT: If you use this option because normal <mode> parameter does not work for you and you use a X server, you'll probably want to set the 'nocrtc' option to ensure that the video mode is properly restored after console <-> X switches.
nocrtc	Do not use CRTC timings while setting the video mode. This option has any effect only if the Video BIOS is VBE 3.0 compliant. Use it if you have problems with modes set the standard way. Note that using this option implies that any refresh rate adjustments will be ignored and the refresh rate will stay at your BIOS default (60 Hz).
noedid	Do not try to fetch and use EDID-provided modes.
noblank	Disable hardware blanking.
v86d:path	Set path to the v86d executable. This option is only available as a module parameter, and not as a part of the video= string. If you need to use it and have uvesafb built into the kernel, use uvesafb.v86d=”path”.

Additionally, the following parameters may be provided. They all override the EDID-provided values and BIOS defaults. Refer to your monitor's specs to get the correct values for maxhf, maxvf and maxclk for your hardware.

maxhf:n	Maximum horizontal frequency (in kHz).
maxvf:n	Maximum vertical frequency (in Hz).
maxclk:n	Maximum pixel clock (in MHz).

34.4 4. The sysfs interface

uvesafb provides several sysfs nodes for configurable parameters and additional information.

Driver attributes:

/sys/bus/platform/drivers/uvesafb

v86d
(default: /sbin/v86d)

Path to the v86d executable. v86d is started by uvesafb if an instance of the daemon isn't already running.

Device attributes:

/sys/bus/platform/drivers/uvesafb/uvesafb.0

nocrtc
Use the default refresh rate (60 Hz) if set to 1.

oem_product_name, oem_product_rev, oem_string, oem_vendor
Information about the card and its maker.

vbe_modes
A list of video modes supported by the Video BIOS along with their VBE mode numbers in hex.

vbe_version
A BCD value indicating the implemented VBE standard.

34.5 5. Miscellaneous

Uvesafb will set a video mode with the default refresh rate and timings from the Video BIOS if you set pixclock to 0 in fb_var_screeninfo.

Michal Januszewski <spock@gentoo.org>

Last updated: 2017-10-10

Documentation of the uvesafb options is loosely based on *What is vesafb?*.

CHAPTER

THIRTYFIVE

WHAT IS VESAFB?

This is a generic driver for a graphic framebuffer on intel boxes.

The idea is simple: Turn on graphics mode at boot time with the help of the BIOS, and use this as framebuffer device /dev/fb0, like the m68k (and other) ports do.

This means we decide at boot time whenever we want to run in text or graphics mode. Switching mode later on (in protected mode) is impossible; BIOS calls work in real mode only. VESA BIOS Extensions Version 2.0 are required, because we need a linear frame buffer.

Advantages:

- It provides a nice large console (128 cols + 48 lines with 1024x768) without using tiny, unreadable fonts.
- You can run XF68_FBDev on top of /dev/fb0 (=> non-accelerated X11 support for every VBE 2.0 compliant graphics board).
- Most important: boot logo :-)

Disadvantages:

- graphic mode is slower than text mode...

35.1 How to use it?

Switching modes is done using the vga=... boot parameter. Read Documentation/admin-guide/svga.rst for details.

You should compile in both vgacon (for text mode) and vesafb (for graphics mode). Which of them takes over the console depends on whenever the specified mode is text or graphics.

The graphic modes are NOT in the list which you get if you boot with vga=ask and hit return. The mode you wish to use is derived from the VESA mode number. Here are those VESA mode numbers:

colors	640x480	800x600	1024x768	1280x1024
256	0x101	0x103	0x105	0x107
32k	0x110	0x113	0x116	0x119
64k	0x111	0x114	0x117	0x11A
16M	0x112	0x115	0x118	0x11B

The video mode number of the Linux kernel is the VESA mode number plus 0x200:

Linux_kernel_mode_number = VESA_mode_number + 0x200

So the table for the Kernel mode numbers are:

colors	640x480	800x600	1024x768	1280x1024
256	0x301	0x303	0x305	0x307
32k	0x310	0x313	0x316	0x319
64k	0x311	0x314	0x317	0x31A
16M	0x312	0x315	0x318	0x31B

To enable one of those modes you have to specify ”vga=ask” in the lilo.conf file and rerun LILO. Then you can type in the desired mode at the ”vga=ask” prompt. For example if you like to use 1024x768x256 colors you have to say ”305” at this prompt.

If this does not work, this might be because your BIOS does not support linear framebuffers or because it does not support this mode at all. Even if your board does, it might be the BIOS which does not. VESA BIOS Extensions v2.0 are required, 1.2 is NOT sufficient. You will get a ”bad mode number” message if something goes wrong.

1. Note: LILO cannot handle hex, for booting directly with ”vga=mode-number” you have to transform the numbers to decimal.
2. Note: Some newer versions of LILO appear to work with those hex values, if you set the 0x in front of the numbers.

35.2 X11

XF68_FBDev should work just fine, but it is non-accelerated. Running another (accelerated) X-Server like XF86_SVGA might or might not work. It depends on X-Server and graphics board.

The X-Server must restore the video mode correctly, else you end up with a broken console (and vesafb cannot do anything about this).

35.3 Refresh rates

There is no way to change the vesafb video mode and/or timings after booting linux. If you are not happy with the 60 Hz refresh rate, you have these options:

- configure and load the DOS-Tools for the graphics board (if available) and boot linux with loadlin.
- use a native driver (matroxfb/atyfb) instead if vesafb. If none is available, write a new one!
- VBE 3.0 might work too. I have neither a gfx board with VBE 3.0 support nor the specs, so I have not checked this yet.

35.4 Configuration

The VESA BIOS provides protected mode interface for changing some parameters. vesafb can use it for palette changes and to pan the display. It is turned off by default because it seems not to work with some BIOS versions, but there are options to turn it on.

You can pass options to vesafb using ”video=vesafb:option” on the kernel command line. Multiple options should be separated by comma, like this: ”video=vesafb:ypan,inverse”

Accepted options:

inverse use inverse color map

ypan	enable display panning using the VESA protected mode interface. The visible screen is just a window of the video memory, console scrolling is done by changing the start of the window. pro: • scrolling (fullscreen) is fast, because there is no need to copy around data. kontra: • scrolling only parts of the screen causes some ugly flicker effects (boot logo flickers for example).
ywrap	Same as ypan, but assumes your gfx board can wrap-around the video memory (i.e. starts reading from top if it reaches the end of video memory). Faster than ypan.
redraw	Scroll by redrawing the affected part of the screen, this is the safe (and slow) default.
vgapal	Use the standard vga registers for palette changes. This is the default.
pmipal	Use the protected mode interface for palette changes.
mtrr:n	Setup memory type range registers for the vesafb framebuffer where n: • 0 - disabled (equivalent to nomtrr) (default) • 1 - uncachable • 2 - write-back • 3 - write-combining • 4 - write-through If you see the following in dmesg, choose the type that matches the old one. In this example, use ”mtrr:2”.
...	
mtrr:	type mismatch for e0000000,8000000 old: write-back new: write-combining
...	
nomtrr	disable mtrr
vremap:n	Remap 'n' MiB of video RAM. If 0 or not specified, remap memory according to video mode. (2.5.66 patch/idea by Antonino Daplas reversed to give override possibility (allocate more fb memory than the kernel would) to 2.4 by tmb@iki.fi)
vtotal:n	If the video BIOS of your card incorrectly determines the total amount of video RAM, use this option to override the BIOS (in MiB).

Have fun!

Gerd Knorr <kraxel@goldbach.in-berlin.de>

Minor (mostly typo) changes by Nico Schmoigl <schmoigl@rumms.uni-mannheim.de>

CHAPTER

THIRTYSIX

VIA INTEGRATION GRAPHIC CHIP CONSOLE FRAMEBUFFER DRIVER

36.1 Platform

The console framebuffer driver is for graphics chips of VIA UniChrome Family (CLE266, PM800 / CN400 / CN300, P4M800CE / P4M800Pro / CN700 / VN800, CX700 / VX700, K8M890, P4M890, CN896 / P4M900, VX800, VX855)

36.2 Driver features

Device: CRT, LCD, DVI

Support viafb_mode:

```
CRT:
    640x480(60, 75, 85, 100, 120 Hz), 720x480(60 Hz),
    720x576(60 Hz), 800x600(60, 75, 85, 100, 120 Hz),
    848x480(60 Hz), 856x480(60 Hz), 1024x512(60 Hz),
    1024x768(60, 75, 85, 100 Hz), 1152x864(75 Hz),
    1280x768(60 Hz), 1280x960(60 Hz), 1280x1024(60, 75, 85 Hz),
    1440x1050(60 Hz), 1600x1200(60, 75 Hz), 1280x720(60 Hz),
    1920x1080(60 Hz), 1400x1050(60 Hz), 800x480(60 Hz)
```

color depth: 8 bpp, 16 bpp, 32 bpp supports.

Support 2D hardware accelerator.

36.3 Using the viafb module

Start viafb with default settings:

```
#modprobe viafb
```

Start viafb with user options:

```
#modprobe viafb viafb_mode=800x600 viafb_bpp=16 viafb_refresh=60
          viafb_active_dev=CRT+DVI viafb_dvi_port=DVP1
```

```
        viafb_mode1=1024x768 viafb_bpp=16 viafb_refresh1=60
        viafb_SAMM_ON=1
```

viafb_mode:

- 640x480 (default)
- 720x480
- 800x600
- 1024x768

viafb_bpp:

- 8, 16, 32 (default:32)

viafb_refresh:

- 60, 75, 85, 100, 120 (default:60)

viafb_lcd_dsp_method:

- 0 : expansion (default)
- 1 : centering

viafb_lcd_mode:
0 : LCD panel with LSB data format input (default) 1 : LCD panel with MSB data format input

viafb_lcd_panel_id:

- 0 : Resolution: 640x480, Channel: single, Dithering: Enable
- 1 : Resolution: 800x600, Channel: single, Dithering: Enable
- 2 : Resolution: 1024x768, Channel: single, Dithering: Enable (default)
- 3 : Resolution: 1280x768, Channel: single, Dithering: Enable
- 4 : Resolution: 1280x1024, Channel: dual, Dithering: Enable
- 5 : Resolution: 1400x1050, Channel: dual, Dithering: Enable
- 6 : Resolution: 1600x1200, Channel: dual, Dithering: Enable
- 8 : Resolution: 800x480, Channel: single, Dithering: Enable
- 9 : Resolution: 1024x768, Channel: dual, Dithering: Enable
- 10: Resolution: 1024x768, Channel: single, Dithering: Disable
- 11: Resolution: 1024x768, Channel: dual, Dithering: Disable
- 12: Resolution: 1280x768, Channel: single, Dithering: Disable
- 13: Resolution: 1280x1024, Channel: dual, Dithering: Disable
- 14: Resolution: 1400x1050, Channel: dual, Dithering: Disable
- 15: Resolution: 1600x1200, Channel: dual, Dithering: Disable
- 16: Resolution: 1366x768, Channel: single, Dithering: Disable
- 17: Resolution: 1024x600, Channel: single, Dithering: Enable

- 18: Resolution: 1280x768, Channel: dual, Dithering: Enable
- 19: Resolution: 1280x800, Channel: single, Dithering: Enable

viafb_accel:

- 0 : No 2D Hardware Acceleration
- 1 : 2D Hardware Acceleration (default)

viafb_SAMM_ON:

- 0 : viafb_SAMM_ON disable (default)
- 1 : viafb_SAMM_ON enable

viafb_mode1: (secondary display device)

- 640x480 (default)
- 720x480
- 800x600
- 1024x768

viafb_bpp1: (secondary display device)

- 8, 16, 32 (default:32)

viafb_refresh1: (secondary display device)

- 60, 75, 85, 100, 120 (default:60)

viafb_active_dev:

This option is used to specify active devices.(CRT, DVI, CRT+LCD...) DVI stands for DVI or HDMI, E.g., If you want to enable HDMI, set viafb_active_dev=DVI. In SAMM case, the previous of viafb_active_dev is primary device, and the following is secondary device.

For example:

To enable one device, such as DVI only, we can use:

```
modprobe viafb viafb_active_dev=DVI
```

To enable two devices, such as CRT+DVI:

```
modprobe viafb viafb_active_dev=CRT+DVI;
```

For DuoView case, we can use:

```
modprobe viafb viafb_active_dev=CRT+DVI
```

OR:

```
modprobe viafb viafb_active_dev=DVI+CRT...
```

For SAMM case:

If CRT is primary and DVI is secondary, we should use:

```
modprobe viafb viafb_active_dev=CRT+DVI viafb_SAMM_ON=1...
```

If DVI is primary and CRT is secondary, we should use:

```
modprobe viafb viafb_active_dev=DVI+CRT viafb_SAMM_ON=1...
```

viafb_display_hardware_layout:
This option is used to specify display hardware layout for CX700 chip.

- 1 : LCD only
- 2 : DVI only
- 3 : LCD+DVI (default)
- 4 : LCD1+LCD2 (internal + internal)
- 16: LCD1+ExternalLCD2 (internal + external)

viafb_second_size:
This option is used to set second device memory size(MB) in SAMM case. The minimal size is 16.

viafb_platform_epia_dvi:
This option is used to enable DVI on EPIA - M

- 0 : No DVI on EPIA - M (default)
- 1 : DVI on EPIA - M

viafb_bus_width:
When using 24 - Bit Bus Width Digital Interface, this option should be set.

- 12: 12-Bit LVDS or 12-Bit TMDS (default)
- 24: 24-Bit LVDS or 24-Bit TMDS

viafb_device_lcd_dualedge:
When using Dual Edge Panel, this option should be set.

- 0 : No Dual Edge Panel (default)
- 1 : Dual Edge Panel

viafb_lcd_port:
This option is used to specify LCD output port, available values are ”DVP0” ”DVP1” ”DFP_HIGHLOW” ”DFP_HIGH” ”DFP_LOW”.

for external LCD + external DVI on CX700(External LCD is on DVP0), we should use:

```
modprobe viafb viafb_lcd_port=DVP0...
```

Notes:

1. CRT may not display properly for DuoView CRT & DVI display at the ”640x480” PAL mode with DVI overscan enabled.
2. SAMM stands for single adapter multi monitors. It is different from multi-head since SAMM support multi monitor at driver layers, thus fbcon layer doesn't even know about it; SAMM's second screen doesn't have a device node file, thus a user mode

application can't access it directly. When SAMM is enabled, viafb_mode and viafb_mode1, viafb_bpp and viafb_bpp1, viafb_refresh and viafb_refresh1 can be different.

3. When console is depending on viafbinfo1, dynamically change resolution and bpp, need to call VIAFB specified ioctl interface VIAFB_SET_DEVICE instead of calling common ioctl function FBIOPUT_VSCREENINFO since viafb doesn't support multi-head well, or it will cause screen crush.

36.4 Configure viafb with ”fbset” tool

”fbset” is an inbox utility of Linux.

1. Inquire current viafb information, type:

```
# fbset -i
```

2. Set various resolutions and viafb_refresh rates:

```
# fbset <resolution-vertical_sync>
```

example:

```
# fbset "1024x768-75"
```

or:

```
# fbset -g 1024 768 1024 768 32
```

Check the file ”/etc/fb.modes” to find display modes available.

3. Set the color depth:

```
# fbset -depth <value>
```

example:

```
# fbset -depth 16
```

36.5 Configure viafb via /proc

The following files exist in /proc/viafb

supported_output_devices
This read-only file contains a full ',' separated list containing all output devices that could be available on your platform. It is likely that not all of those have a connector on your hardware but it should provide a good starting point to figure out which of those names match a real connector.

Example:

```
# cat /proc/viafb/supported_output_devices
```

iga1/output_devices, iga2/output_devices

These two files are readable and writable. iga1 and iga2 are the two independent units that produce the screen image. Those images can be forwarded to one or more output devices. Reading those files is a way to query which output devices are currently used by an iga.

Example:

```
# cat /proc/viafb/iga1/output_devices
```

If there are no output devices printed the output of this iga is lost. This can happen for example if only one (the other) iga is used. Writing to these files allows adjusting the output devices during runtime. One can add new devices, remove existing ones or switch between igas. Essentially you can write a ',' separated list of device names (or a single one) in the same format as the output to those files. You can add a '+' or '-' as a prefix allowing simple addition and removal of devices. So a prefix '+' adds the devices from your list to the already existing ones, '-' removes the listed devices from the existing ones and if no prefix is given it replaces all existing ones with the listed ones. If you remove devices they are expected to turn off. If you add devices that are already part of the other iga they are removed there and added to the new one.

Examples:

Add CRT as output device to iga1:

```
# echo +CRT > /proc/viafb/iga1/output_devices
```

Remove (turn off) DVP1 and LVDS1 as output devices of iga2:

```
# echo -DVP1,LVDS1 > /proc/viafb/iga2/output_devices
```

Replace all iga1 output devices by CRT:

```
# echo CRT > /proc/viafb/iga1/output_devices
```

36.6 Bootup with viafb

Add the following line to your grub.conf:

```
append = "video=viafb:viafb_mode=1024x768,viafb_bpp=32,viafb_refresh=85"
```

36.6.1 VIA Framebuffer modes

```
#
#
#   These data are based on the CRTC parameters in
#
#       VIA Integration Graphics Chip
#       (C) 2004 VIA Technologies Inc.
#


#
#   640x480, 60 Hz, Non-Interlaced (25.175 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      640     480
#   Scan Frequency      31.469 kHz  59.94 Hz
#   Sync Width      3.813 us    0.064 ms
#               12 chars    2 lines
#   Front Porch     0.636 us    0.318  ms
#               2 chars     10 lines
#   Back Porch      1.907 us    1.048  ms
#               6 chars     33 lines
#   Active Time     25.422 us   15.253 ms
#               80 chars    480 lines
#   Blank Time      6.356 us    1.430 ms
#               20 chars    45 lines
#   Polarity        negative    negative
#

mode "640x480-60"
# D: 25.175 MHz, H: 31.469 kHz, V: 59.94 Hz
    geometry 640 480 640 480 32
    timings 39722 48 16 33 10 96 2 endmode mode "480x640-60"
# D: 24.823 MHz, H: 39.780 kHz, V: 60.00 Hz
    geometry 480 640 480 640 32 timings 39722 72 24 19 1 48 3 endmode
#
#   640x480, 75 Hz, Non-Interlaced (31.50 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      640     480
#   Scan Frequency      37.500 kHz  75.00 Hz
#   Sync Width      2.032 us    0.080 ms
#               8 chars     3 lines
#   Front Porch     0.508 us    0.027 ms
#               2 chars     1 lines
#   Back Porch      3.810 us    0.427 ms
#               15 chars    16 lines
#   Active Time     20.317 us   12.800 ms
#               80 chars    480 lines
#   Blank Time      6.349 us    0.533 ms
#               25 chars    20 lines
```

```
#   Polarity        negative    negative
#
    mode "640x480-75"
# D: 31.50 MHz, H: 37.500 kHz, V: 75.00 Hz
    geometry 640 480 640 480 32 timings 31747 120 16 16 1 64 3 endmode
#
#   640x480, 85 Hz, Non-Interlaced (36.000 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      640     480
#   Scan Frequency  43.269 kHz  85.00 Hz
#   Sync Width      1.556 us    0.069 ms
#               7 chars     3 lines
#   Front Porch     1.556 us    0.023 ms
#               7 chars     1 lines
#   Back Porch      2.222 us    0.578 ms
#               10 chars    25 lines
#   Active Time     17.778 us   11.093 ms
#               80 chars    480 lines
#   Blank Time      5.333 us    0.670 ms
#               24 chars    29 lines
#   Polarity        negative    negative
#
    mode "640x480-85"
# D: 36.000 MHz, H: 43.269 kHz, V: 85.00 Hz
    geometry 640 480 640 480 32 timings 27777 80 56 25 1 56 3 endmode
#
#   640x480, 100 Hz, Non-Interlaced (43.163 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      640     480
#   Scan Frequency      50.900 kHz  100.00 Hz
#   Sync Width      1.483 us    0.058 ms
#               8 chars     3 lines
#   Front Porch     0.927 us    0.019 ms
#               5 chars     1 lines
#   Back Porch      2.409 us    0.475 ms
#               13 chars    25 lines
#   Active Time     14.827 us   9.430 ms
#               80 chars    480 lines
#   Blank Time      4.819 us    0.570 ms
#               26 chars    29 lines
#   Polarity        positive    positive
#
    mode "640x480-100"
# D: 43.163 MHz, H: 50.900 kHz, V: 100.00 Hz
    geometry 640 480 640 480 32 timings 23168 104 40 25 1 64 3 endmode
#
#   640x480, 120 Hz, Non-Interlaced (52.406 MHz dotclock)
#
```

```
#               Horizontal  Vertical
#   Resolution      640     480
#   Scan Frequency      61.800 kHz  120.00 Hz
#   Sync Width      1.221 us    0.048 ms
#               8 chars         3 lines
#   Front Porch     0.763 us    0.016 ms
#               5 chars     1 lines
#   Back Porch      1.984 us    0.496 ms
#               13 chars    31 lines
#   Active Time     12.212 us   7.767 ms
#               80 chars    480 lines
#   Blank Time      3.969 us    0.566 ms
#               26 chars    35 lines
#   Polarity        positive    positive
#
    mode "640x480-120"
# D: 52.406 MHz, H: 61.800 kHz, V: 120.00 Hz
    geometry 640 480 640 480 32 timings 19081 104 40 31 1 64 3 endmode
#
#   720x480, 60 Hz, Non-Interlaced (26.880 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      720     480
#   Scan Frequency      30.000 kHz  60.241 Hz
#   Sync Width      2.679 us    0.099 ms
#               9 chars     3 lines
#   Front Porch     0.595 us    0.033 ms
#               2 chars     1 lines
#   Back Porch      3.274 us    0.462 ms
#               11 chars    14 lines
#   Active Time     26.786 us   16.000 ms
#               90 chars    480 lines
#   Blank Time      6.548 us    0.600 ms
#               22 chars    18 lines
#   Polarity        positive    positive
#
    mode "720x480-60"
# D: 26.880 MHz, H: 30.000 kHz, V: 60.24 Hz
    geometry 720 480 720 480 32 timings 37202 88 16 14 1 72 3 endmode
#
#   800x480, 60 Hz, Non-Interlaced (29.581 MHz dotclock)
#
#               Horizontal    Vertical
#   Resolution      800         480
#   Scan Frequency  29.892 kHz  60.00 Hz
#   Sync Width      2.704 us    100.604 us
#                   10 chars    3 lines
#   Front Porch     0.541 us    33.535 us
#                   2 chars     1 lines
#   Back Porch      3.245 us    435.949 us
```

```
#                   12 chars    13 lines
#   Active Time     27.044 us   16.097 ms
#                   100 chars   480 lines
#   Blank Time      6.491 us    0.570 ms
#                   24 chars    17 lines
#   Polarity        positive    positive
#
    mode "800x480-60"
# D: 29.500 MHz, H: 29.738 kHz, V: 60.00 Hz
    geometry 800 480 800 480 32 timings 33805 96 24 10 3 72 7 endmode
#
#   720x576, 60 Hz, Non-Interlaced (32.668 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      720     576
#   Scan Frequency      35.820 kHz  60.00 Hz
#   Sync Width      2.204 us    0.083 ms
#               9 chars     3 lines
#   Front Porch     0.735 us    0.027 ms
#               3 chars     1 lines
#   Back Porch      2.939 us    0.459 ms
#               12 chars    17 lines
#   Active Time     22.040 us   16.080 ms
#               90 chars    476 lines
#   Blank Time      5.877 us    0.586 ms
#               24 chars    21 lines
#   Polarity        positive    positive
#
    mode "720x576-60"
# D: 32.668 MHz, H: 35.820 kHz, V: 60.00 Hz
    geometry 720 576 720 576 32 timings 30611 96 24 17 1 72 3 endmode
#
#   800x600, 60 Hz, Non-Interlaced (40.00 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      800     600
#   Scan Frequency      37.879 kHz  60.32 Hz
#   Sync Width      3.200 us    0.106 ms
#               16 chars    4 lines
#   Front Porch     1.000 us    0.026 ms
#               5 chars     1 lines
#   Back Porch      2.200 us    0.607 ms
#               11 chars    23 lines
#   Active Time     20.000 us   15.840 ms
#               100 chars   600 lines
#   Blank Time      6.400 us    0.739 ms
#               32 chars    28 lines
#   Polarity        positive    positive
#
    mode "800x600-60"
```

```
# D: 40.00 MHz, H: 37.879 kHz, V: 60.32 Hz
    geometry 800 600 800 600 32
    timings 25000 88 40 23 1 128 4 hsync high vsync high endmode
#
#   800x600, 75 Hz, Non-Interlaced (49.50 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      800     600
#   Scan Frequency      46.875 kHz  75.00 Hz
#   Sync Width      1.616 us    0.064 ms
#               10 chars    3 lines
#   Front Porch     0.323 us    0.021 ms
#               2 chars     1 lines
#   Back Porch      3.232 us    0.448 ms
#               20 chars    21 lines
#   Active Time     16.162 us   12.800 ms
#               100 chars   600 lines
#   Blank Time      5.172 us    0.533 ms
#               32 chars    25 lines
#   Polarity        positive    positive
#
    mode "800x600-75"
# D: 49.50 MHz, H: 46.875 kHz, V: 75.00 Hz
    geometry 800 600 800 600 32
    timings 20203 160 16 21 1 80 3 hsync high vsync high endmode
#
#   800x600, 85 Hz, Non-Interlaced (56.25 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      800     600
#   Scan Frequency      53.674 kHz  85.061 Hz
#   Sync Width      1.138 us    0.056 ms
#               8 chars     3 lines
#   Front Porch     0.569 us    0.019 ms
#               4 chars     1 lines
#   Back Porch      2.702 us    0.503 ms
#               19 chars    27 lines
#   Active Time     14.222 us   11.179 ms
#               100 chars   600 lines
#   Blank Time      4.409 us    0.578 ms
#               31 chars    31 lines
#   Polarity        positive    positive
#
    mode "800x600-85"
# D: 56.25 MHz, H: 53.674 kHz, V: 85.061 Hz
    geometry 800 600 800 600 32
    timings 17777 152 32 27 1 64 3 hsync high vsync high endmode
#
#   800x600, 100 Hz, Non-Interlaced (67.50 MHz dotclock)
#
```

```
#               Horizontal  Vertical
#   Resolution      800     600
#   Scan Frequency      62.500 kHz  100.00 Hz
#   Sync Width      0.948 us    0.064 ms
#               8 chars     4 lines
#   Front Porch     0.000 us    0.112 ms
#               0 chars     7 lines
#   Back Porch      3.200 us    0.224 ms
#               27 chars    14 lines
#   Active Time     11.852 us   9.600 ms
#               100 chars   600 lines
#   Blank Time      4.148 us    0.400 ms
#               35 chars    25 lines
#   Polarity        positive    positive
#
    mode "800x600-100"
# D: 67.50 MHz, H: 62.500 kHz, V: 100.00 Hz
    geometry 800 600 800 600 32
    timings 14667 216 0 14 7 64 4 hsync high vsync high endmode
#
#   800x600, 120 Hz, Non-Interlaced (83.950 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      800     600
#   Scan Frequency  77.160 kHz  120.00 Hz
#   Sync Width      1.048 us    0.039 ms
#               11 chars    3 lines
#   Front Porch     0.667 us    0.013 ms
#               7 chars     1 lines
#   Back Porch      1.715 us    0.507 ms
#               18 chars    39 lines
#   Active Time     9.529 us    7.776 ms
#               100 chars   600 lines
#   Blank Time      3.431 us    0.557 ms
#               36 chars    43 lines
#   Polarity        positive    positive
#
    mode "800x600-120"
# D: 83.950 MHz, H: 77.160 kHz, V: 120.00 Hz
    geometry 800 600 800 600 32
    timings 11912 144 56 39 1 88 3 hsync high vsync high endmode
#
#   848x480, 60 Hz, Non-Interlaced (31.490 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      848     480
#   Scan Frequency  29.820 kHz  60.00 Hz
#   Sync Width      2.795 us    0.099 ms
#               11 chars    3 lines
#   Front Porch     0.508 us    0.033 ms
```

```
#               2 chars     1 lines
#   Back Porch      3.303 us    0.429 ms
#               13 chars    13 lines
#   Active Time     26.929 us   16.097 ms
#               106 chars   480 lines
#   Blank Time      6.605 us    0.570 ms
#               26 chars    17 lines
#   Polarity        positive    positive
#
    mode "848x480-60"
# D: 31.500 MHz, H: 29.830 kHz, V: 60.00 Hz
    geometry 848 480 848 480 32
    timings 31746 104 24 12 3 80 5 hsync high vsync high endmode
#
#   856x480, 60 Hz, Non-Interlaced (31.728 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      856     480
#   Scan Frequency  29.820 kHz  60.00 Hz
#   Sync Width      2.774 us    0.099 ms
#               11 chars    3 lines
#   Front Porch     0.504 us    0.033 ms
#               2 chars     1 lines
#   Back Porch      3.728 us    0.429 ms
#               13 chars    13 lines
#   Active Time     26.979 us   16.097 ms
#               107 chars   480 lines
#   Blank Time      6.556 us    0.570 ms
#               26 chars    17 lines
#   Polarity        positive    positive
#
    mode "856x480-60"
# D: 31.728 MHz, H: 29.820 kHz, V: 60.00 Hz
    geometry 856 480 856 480 32
    timings 31518 104 16 13 1 88 3
    hsync high vsync high endmode mode "960x600-60"
# D: 45.250 MHz, H: 37.212 kHz, V: 60.00 Hz
    geometry 960 600 960 600 32 timings 22099 128 32 15 3 96 6 endmode
#
#   1000x600, 60 Hz, Non-Interlaced (48.068 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1000     600
#   Scan Frequency  37.320 kHz  60.00 Hz
#   Sync Width      2.164 us    0.080 ms
#               13 chars    3 lines
#   Front Porch     0.832 us    0.027 ms
#               5 chars     1 lines
#   Back Porch      2.996 us    0.483 ms
#               18 chars    18 lines
```

```
#   Active Time     20.804 us   16.077 ms
#               125 chars   600 lines
#   Blank Time      5.991 us    0.589 ms
#               36 chars    22 lines
#   Polarity        negative    positive
#
    mode "1000x600-60"
# D: 48.068 MHz, H: 37.320 kHz, V: 60.00 Hz
    geometry 1000 600 1000 600 32
    timings 20834 144 40 18 1 104 3 endmode mode "1024x576-60"
# D: 46.996 MHz, H: 35.820 kHz, V: 60.00 Hz
    geometry 1024 576 1024 576 32
    timings 21278 144 40 17 1 104 3 endmode mode "1024x600-60"
# D: 48.964 MHz, H: 37.320 kHz, V: 60.00 Hz
    geometry 1024 600 1024 600 32
    timings 20461 144 40 18 1 104 3 endmode mode "1088x612-60"
# D: 52.952 MHz, H: 38.040 kHz, V: 60.00 Hz
    geometry 1088 612 1088 612 32 timings 18877 152 48 16 3 104 5 endmode
#
#   1024x512, 60 Hz, Non-Interlaced (41.291 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1024    512
#   Scan Frequency  31.860 kHz  60.00 Hz
#   Sync Width      2.519 us    0.094 ms
#               13 chars    3 lines
#   Front Porch     0.775 us    0.031 ms
#               4 chars     1 lines
#   Back Porch      3.294 us    0.465 ms
#               17 chars    15 lines
#   Active Time     24.800 us   16.070 ms
#               128 chars   512 lines
#   Blank Time      6.587 us    0.596 ms
#               34 chars    19 lines
#   Polarity        positive    positive
#
    mode "1024x512-60"
# D: 41.291 MHz, H: 31.860 kHz, V: 60.00 Hz
    geometry 1024 512 1024 512 32
    timings 24218 126 32 15 1 104 3 hsync high vsync high endmode
#
#   1024x600, 60 Hz, Non-Interlaced (48.875 MHz dotclock)
#
#                     Horizontal  Vertical
#   Resolution          1024        768
#   Scan Frequency      37.252 kHz  60.00 Hz
#   Sync Width          2.128 us    80.532us
#                       13 chars    3 lines
#   Front Porch        0.818 us     26.844 us
#                       5 chars     1 lines
```

```
#   Back Porch          2.946 us    483.192 us
#                       18 chars    18 lines
#   Active Time         20.951 us   16.697 ms
#                       128 chars   622 lines
#   Blank Time          5.893 us    0.591 ms
#                       36 chars    22 lines
#   Polarity            negative    positive
#
#mode "1024x600-60"
#     # D: 48.875 MHz, H: 37.252 kHz, V: 60.00 Hz
#     geometry 1024 600 1024 600 32
#    timings 20460  144 40 18 1 104  3
# endmode
#
#   1024x768, 60 Hz, Non-Interlaced (65.00 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1024        768
#   Scan Frequency      48.363 kHz  60.00 Hz
#   Sync Width      2.092 us    0.124 ms
#               17 chars    6 lines
#   Front Porch     0.369 us    0.062 ms
#               3 chars     3 lines
#   Back Porch      2.462 us    0.601 ms
#               20 chars    29 lines
#   Active Time     15.754 us   15.880 ms
#               128 chars   768 lines
#   Blank Time      4.923 us    0.786 ms
#               40 chars    38 lines
#   Polarity        negative    negative
#
    mode "1024x768-60"
# D: 65.00 MHz, H: 48.363 kHz, V: 60.00 Hz
    geometry 1024 768 1024 768 32 timings 15385 160 24 29 3 136 6 endmode
#
#   1024x768, 75 Hz, Non-Interlaced (78.75 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1024        768
#   Scan Frequency      60.023 kHz  75.03 Hz
#   Sync Width      1.219 us    0.050 ms
#               12 chars    3 lines
#   Front Porch     0.203 us    0.017 ms
#               2 chars     1 lines
#   Back Porch      2.235 us    0.466 ms
#               22 chars    28 lines
#   Active Time     13.003 us   12.795 ms
#               128 chars   768 lines
#   Blank Time      3.657 us    0.533 ms
#               36 chars    32 lines
```

```
#   Polarity        positive    positive
#
    mode "1024x768-75"
# D: 78.75 MHz, H: 60.023 kHz, V: 75.03 Hz
    geometry 1024 768 1024 768 32
    timings 12699 176 16 28 1 96 3 hsync high vsync high endmode
#
#   1024x768, 85 Hz, Non-Interlaced (94.50 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1024        768
#   Scan Frequency  68.677 kHz  85.00 Hz
#   Sync Width      1.016 us    0.044 ms
#               12 chars    3 lines
#   Front Porch     0.508 us    0.015 ms
#               6 chars     1 lines
#   Back Porch      2.201 us    0.524 ms
#               26 chars    36 lines
#   Active Time     10.836 us   11.183 ms
#               128 chars   768 lines
#   Blank Time      3.725 us    0.582 ms
#               44 chars    40 lines
#   Polarity        positive    positive
#
    mode "1024x768-85"
# D: 94.50 MHz, H: 68.677 kHz, V: 85.00 Hz
    geometry 1024 768 1024 768 32
    timings 10582 208 48 36 1 96 3 hsync high vsync high endmode
#
#   1024x768, 100 Hz, Non-Interlaced (110.0 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1024        768
#   Scan Frequency      79.023 kHz  99.78 Hz
#   Sync Width      0.800 us    0.101 ms
#               11 chars    8 lines
#   Front Porch     0.000 us    0.000 ms
#               0 chars     0 lines
#   Back Porch      2.545 us    0.202 ms
#               35 chars    16 lines
#   Active Time     9.309 us    9.719 ms
#               128 chars   768 lines
#   Blank Time      3.345 us    0.304 ms
#               46 chars    24 lines
#   Polarity        negative    negative
#
    mode "1024x768-100"
# D: 113.3 MHz, H: 79.023 kHz, V: 99.78 Hz
    geometry 1024 768 1024 768 32
    timings 8825 280 0 16 0 88 8 endmode mode "1152x720-60"
```

```
# D: 66.750 MHz, H: 44.859 kHz, V: 60.00 Hz
    geometry 1152 720 1152 720 32 timings 14981 168 56 19 3 112 6 endmode
#
#   1152x864, 75 Hz, Non-Interlaced (110.0 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1152        864
#   Scan Frequency      75.137 kHz  74.99 Hz
#   Sync Width      1.309 us    0.106 ms
#               18 chars    8 lines
#   Front Porch     0.245 us    0.599 ms
#               3 chars     45 lines
#   Back Porch      1.282 us    1.132 ms
#               18 chars    85 lines
#   Active Time     10.473 us   11.499 ms
#               144 chars   864 lines
#   Blank Time      2.836 us    1.837 ms
#               39 chars    138 lines
#   Polarity        positive    positive
#
    mode "1152x864-75"
# D: 110.0 MHz, H: 75.137 kHz, V: 74.99 Hz
    geometry 1152 864 1152 864 32
    timings 9259 144 24 85 45 144 8
    hsync high vsync high endmode mode "1200x720-60"
# D: 70.184 MHz, H: 44.760 kHz, V: 60.00 Hz
    geometry 1200 720 1200 720 32
    timings 14253 184 28 22 1 128 3 endmode mode "1280x600-60"
# D: 61.503 MHz, H: 37.320 kHz, V: 60.00 Hz
    geometry 1280 600 1280 600 32
    timings 16260 184 28 18 1 128 3 endmode mode "1280x720-50"
# D: 60.466 MHz, H: 37.050 kHz, V: 50.00 Hz
    geometry 1280 720 1280 720 32
    timings 16538 176 48 17 1 128 3 endmode mode "1280x768-50"
# D: 65.178 MHz, H: 39.550 kHz, V: 50.00 Hz
    geometry 1280 768 1280 768 32 timings 15342 184 28 19 1 128 3 endmode
#
#   1280x768, 60 Hz, Non-Interlaced (80.136 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280    768
#   Scan Frequency  47.700 kHz  60.00 Hz
#   Sync Width      1.697 us    0.063 ms
#               17 chars    3 lines
#   Front Porch     0.799 us    0.021 ms
#               8 chars     1 lines
#   Back Porch      2.496 us    0.483 ms
#               25 chars    23 lines
#   Active Time     15.973 us   16.101 ms
#               160 chars   768 lines
```

```
#   Blank Time      4.992 us    0.566 ms
#               50 chars    27 lines
#   Polarity        positive    positive
#
    mode "1280x768-60"
# D: 80.13 MHz, H: 47.700 kHz, V: 60.00 Hz
    geometry 1280 768 1280 768 32
    timings 12480 200 48 23 1 126 3 hsync high vsync high endmode
#
#   1280x800, 60 Hz, Non-Interlaced (83.375 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280    800
#   Scan Frequency  49.628 kHz  60.00 Hz
#   Sync Width      1.631 us    60.450 us
#                   17 chars    3 lines
#   Front Porch     0.768 us    20.15 us
#                   8 chars     1 lines
#   Back Porch      2.399 us    0.483 ms
#                   25 chars    24 lines
#   Active Time     15.352 us   16.120 ms
#                   160 chars   800 lines
#   Blank Time      4.798 us    0.564 ms
#                   50 chars    28 lines
#   Polarity        negative    positive
#
    mode "1280x800-60"
# D: 83.500 MHz, H: 49.702 kHz, V: 60.00 Hz
    geometry 1280 800 1280 800 32 timings 11994 200 72 22 3 128 6 endmode
#
#   1280x960, 60 Hz, Non-Interlaced (108.00 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280    960
#   Scan Frequency  60.000 kHz  60.00 Hz
#   Sync Width      1.037 us    0.050 ms
#               14 chars    3 lines
#   Front Porch     0.889 us    0.017 ms
#               12 chars    1 lines
#   Back Porch      2.889 us    0.600 ms
#               39 chars    36 lines
#   Active Time     11.852 us   16.000 ms
#               160 chars   960 lines
#   Blank Time      4.815 us    0.667 ms
#               65 chars    40 lines
#   Polarity        positive    positive
#
    mode "1280x960-60"
# D: 108.00 MHz, H: 60.000 kHz, V: 60.00 Hz
    geometry 1280 960 1280 960 32
```

```
    timings 9259 312 96 36 1 112 3 hsync high vsync high endmode
#
#   1280x1024, 60 Hz, Non-Interlaced (108.00 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280        1024
#   Scan Frequency      63.981 kHz  60.02 Hz
#   Sync Width      1.037 us    0.047 ms
#               14 chars    3 lines
#   Front Porch     0.444 us    0.015 ms
#               6 chars     1 lines
#   Back Porch      2.297 us    0.594 ms
#               31 chars    38 lines
#   Active Time     11.852 us   16.005 ms
#               160 chars   1024 lines
#   Blank Time      3.778 us    0.656 ms
#               51 chars    42 lines
#   Polarity        positive    positive
#
    mode "1280x1024-60"
# D: 108.00 MHz, H: 63.981 kHz, V: 60.02 Hz
    geometry 1280 1024 1280 1024 32
    timings 9260 248 48 38 1 112 3 hsync high vsync high endmode
#
#   1280x1024, 75 Hz, Non-Interlaced (135.00 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280        1024
#   Scan Frequency      79.976 kHz  75.02 Hz
#   Sync Width      1.067 us    0.038 ms
#               18 chars    3 lines
#   Front Porch     0.119 us    0.012 ms
#               2 chars     1 lines
#   Back Porch      1.837 us    0.475 ms
#               31 chars    38 lines
#   Active Time     9.481 us    12.804 ms
#               160 chars   1024 lines
#   Blank Time      3.022 us    0.525 ms
#               51 chars    42 lines
#   Polarity        positive    positive
#
    mode "1280x1024-75"
# D: 135.00 MHz, H: 79.976 kHz, V: 75.02 Hz
    geometry 1280 1024 1280 1024 32
    timings 7408 248 16 38 1 144 3 hsync high vsync high endmode
#
#   1280x1024, 85 Hz, Non-Interlaced (157.50 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280        1024
```

```
#   Scan Frequency  91.146 kHz  85.02 Hz
#   Sync Width      1.016 us    0.033 ms
#               20 chars    3 lines
#   Front Porch     0.406 us    0.011 ms
#               8 chars     1 lines
#   Back Porch      1.422 us    0.483 ms
#               28 chars    44 lines
#   Active Time     8.127 us    11.235 ms
#               160 chars   1024 lines
#   Blank Time      2.844 us    0.527 ms
#               56 chars    48 lines
#   Polarity        positive    positive
#
    mode "1280x1024-85"
# D: 157.50 MHz, H: 91.146 kHz, V: 85.02 Hz
    geometry 1280 1024 1280 1024 32
    timings 6349 224 64 44 1 160 3
    hsync high vsync high endmode mode "1440x900-60"
# D: 106.500 MHz, H: 55.935 kHz, V: 60.00 Hz
    geometry 1440 900 1440 900 32
    timings 9390 232 80 25 3 152 6
    hsync high vsync high endmode mode "1440x900-75"
# D: 136.750 MHz, H: 70.635 kHz, V: 75.00 Hz
    geometry 1440 900 1440 900 32
    timings 7315 248 96 33 3 152 6 hsync high vsync high endmode
#
#   1440x1050, 60 Hz, Non-Interlaced (125.10 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1440        1050
#   Scan Frequency      65.220 kHz  60.00 Hz
#   Sync Width      1.204 us    0.046 ms
#               19 chars    3 lines
#   Front Porch     0.760 us    0.015 ms
#               12 chars    1 lines
#   Back Porch      1.964 us    0.495 ms
#               31 chars    33 lines
#   Active Time     11.405 us   16.099 ms
#               180 chars   1050 lines
#   Blank Time      3.928 us    0.567 ms
#               62 chars    37 lines
#   Polarity        positive    positive
#
    mode "1440x1050-60"
# D: 125.10 MHz, H: 65.220 kHz, V: 60.00 Hz
    geometry 1440 1050 1440 1050 32
    timings 7993 248 96 33 1 152 3
    hsync high vsync high endmode mode "1600x900-60"
# D: 118.250 MHz, H: 55.990 kHz, V: 60.00 Hz
    geometry 1600 900 1600 900 32
```

```
    timings 8415 256 88 26 3 168 5 endmode mode "1600x1024-60"
# D: 136.358 MHz, H: 63.600 kHz, V: 60.00 Hz
    geometry 1600 1024 1600 1024 32 timings 7315 272 104 32 1 168 3 endmode
#
#   1600x1200, 60 Hz, Non-Interlaced (156.00 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1600        1200
#   Scan Frequency      76.200 kHz  60.00 Hz
#   Sync Width      1.026 us    0.105 ms
#               20 chars    8 lines
#   Front Porch     0.205 us    0.131 ms
#               4 chars     10 lines
#   Back Porch      1.636 us    0.682 ms
#               32 chars    52 lines
#   Active Time     10.256 us   15.748 ms
#               200 chars   1200 lines
#   Blank Time      2.872 us    0.866 ms
#               56 chars    66 lines
#   Polarity        negative    negative
#
    mode "1600x1200-60"
# D: 156.00 MHz, H: 76.200 kHz, V: 60.00 Hz
    geometry 1600 1200 1600 1200 32 timings 6172 256 32 52 10 160 8 endmode
#
#   1600x1200, 75 Hz, Non-Interlaced (202.50 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1600        1200
#   Scan Frequency  93.750 kHz  75.00 Hz
#   Sync Width      0.948 us    0.032 ms
#               24 chars    3 lines
#   Front Porch     0.316 us    0.011 ms
#               8 chars     1 lines
#   Back Porch      1.501 us    0.491 ms
#               38 chars    46 lines
#   Active Time     7.901 us    12.800 ms
#               200 chars   1200 lines
#   Blank Time      2.765 us    0.533 ms
#               70 chars    50 lines
#   Polarity    positive    positive
#
    mode "1600x1200-75"
# D: 202.50 MHz, H: 93.750 kHz, V: 75.00 Hz
    geometry 1600 1200 1600 1200 32
    timings 4938 304 64 46 1 192 3
    hsync high vsync high endmode mode "1680x1050-60"
# D: 146.250 MHz, H: 65.290 kHz, V: 59.954 Hz
    geometry 1680 1050 1680 1050 32
    timings 6814 280 104 30 3 176 6
```

```
    hsync high vsync high endmode mode "1680x1050-75"
# D: 187.000 MHz, H: 82.306 kHz, V: 74.892 Hz
    geometry 1680 1050 1680 1050 32
    timings 5348 296 120 40 3 176 6
    hsync high vsync high endmode mode "1792x1344-60"
# D: 202.975 MHz, H: 83.460 kHz, V: 60.00 Hz
    geometry 1792 1344 1792 1344 32
    timings 4902 320 128 43 1 192 3
    hsync high vsync high endmode mode "1856x1392-60"
# D: 218.571 MHz, H: 86.460 kHz, V: 60.00 Hz
    geometry 1856 1392 1856 1392 32
    timings 4577 336 136 45 1 200 3
    hsync high vsync high endmode mode "1920x1200-60"
# D: 193.250 MHz, H: 74.556 kHz, V: 60.00 Hz
    geometry 1920 1200 1920 1200 32
    timings 5173 336 136 36 3 200 6
    hsync high vsync high endmode mode "1920x1440-60"
# D: 234.000 MHz, H:90.000 kHz, V: 60.00 Hz
    geometry 1920 1440 1920 1440 32
    timings 4274 344 128 56 1 208 3
    hsync high vsync high endmode mode "1920x1440-75"
# D: 297.000 MHz, H:112.500 kHz, V: 75.00 Hz
    geometry 1920 1440 1920 1440 32
    timings 3367 352 144 56 1 224 3
    hsync high vsync high endmode mode "2048x1536-60"
# D: 267.250 MHz, H: 95.446 kHz, V: 60.00 Hz
    geometry 2048 1536 2048 1536 32
    timings 3742 376 152 49 3 224 4 hsync high vsync high endmode
#
#   1280x720, 60 Hz, Non-Interlaced (74.481 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1280        720
#   Scan Frequency      44.760 kHz  60.00 Hz
#   Sync Width      1.826 us    67.024 ms
#               17 chars    3 lines
#   Front Porch     0.752 us    22.341 ms
#               7 chars     1 lines
#   Back Porch      2.578 us    491.510 ms
#               24 chars    22 lines
#   Active Time     17.186 us   16.086 ms
#               160 chars   720 lines
#   Blank Time      5.156 us    0.581 ms
#               48 chars    26 lines
#   Polarity        negative    negative
#
    mode "1280x720-60"
# D: 74.481 MHz, H: 44.760 kHz, V: 60.00 Hz
    geometry 1280 720 1280 720 32 timings 13426 192 64 22 1 136 3 endmode
#
```

```
#   1920x1080, 60 Hz, Non-Interlaced (172.798 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1920        1080
#   Scan Frequency      67.080 kHz  60.00 Hz
#   Sync Width      1.204 us    44.723 ms
#               26 chars    3 lines
#   Front Porch     0.694 us    14.908 ms
#               15 chars     1 lines
#   Back Porch      1.898 us    506.857 ms
#               41 chars    34 lines
#   Active Time     11.111 us   16.100 ms
#               240 chars   1080 lines
#   Blank Time      3.796 us    0.566 ms
#               82 chars    38 lines
#   Polarity        negative    negative
#
    mode "1920x1080-60"
# D: 74.481 MHz, H: 67.080 kHz, V: 60.00 Hz
    geometry 1920 1080 1920 1080 32 timings 5787 328 120 34 1 208 3 endmode
#
#   1400x1050, 60 Hz, Non-Interlaced (122.61 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1400        1050
#   Scan Frequency      65.218 kHz  59.99 Hz
#   Sync Width      1.037 us    0.047 ms
#               19 chars    3 lines
#   Front Porch     0.444 us    0.015 ms
#               11 chars     1 lines
#   Back Porch      1.185 us    0.188 ms
#               30 chars    33 lines
#   Active Time     12.963 us   16.411 ms
#               175 chars   1050 lines
#   Blank Time      2.667 us    0.250 ms
#               60 chars    37 lines
#   Polarity        negative    positive
#
    mode "1400x1050-60"
# D: 122.750 MHz, H: 65.317 kHz, V: 59.99 Hz
    geometry 1400 1050 1408 1050 32
    timings 8214 232 88 32 3 144 4 endmode mode "1400x1050-75"
# D: 156.000 MHz, H: 82.278 kHz, V: 74.867 Hz
    geometry 1400 1050 1408 1050 32 timings 6410 248 104 42 3 144 4 endmode
#
#   1366x768, 60 Hz, Non-Interlaced (85.86 MHz dotclock)
#
#               Horizontal  Vertical
#   Resolution      1366        768
#   Scan Frequency      47.700 kHz  60.00 Hz
```

```
#   Sync Width      1.677 us    0.063 ms
#               18 chars    3 lines
#   Front Porch     0.839 us    0.021 ms
#               9 chars     1 lines
#   Back Porch      2.516 us    0.482 ms
#               27 chars    23 lines
#   Active Time     15.933 us   16.101 ms
#               171 chars   768 lines
#   Blank Time      5.031 us    0.566 ms
#               54 chars    27 lines
#   Polarity        negative    positive
#
    mode "1360x768-60"
# D: 84.750 MHz, H: 47.720 kHz, V: 60.00 Hz
    geometry 1360 768 1360 768 32
    timings 11799 208 72 22 3 136 5 endmode mode "1366x768-60"
# D: 85.86 MHz, H: 47.700 kHz, V: 60.00 Hz
    geometry 1366 768 1366 768 32
    timings 11647 216 72 23 1 144 3 endmode mode "1366x768-50"
# D: 69,924 MHz, H: 39.550 kHz, V: 50.00 Hz
    geometry 1366 768 1366 768 32 timings 14301 200 56 19 1 144 3 endmode
```

CHAPTER

THIRTYSEVEN

VT8623FB - FBDEV DRIVER FOR GRAPHICS CORE IN VIA VT8623 CHIPSET

37.1 Supported Hardware

VIA VT8623 [CLE266] chipset and its graphics core (known as CastleRock or Unichrome)

I tested vt8623fb on VIA EPIA ML-6000

37.2 Supported Features

- 4 bpp pseudocolor modes (with 18bit palette, two variants)
- 8 bpp pseudocolor mode (with 18bit palette)
- 16 bpp truecolor mode (RGB 565)
- 32 bpp truecolor mode (RGB 888)
- text mode (activated by bpp = 0)
- doublescan mode variant (not available in text mode)
- panning in both directions
- suspend/resume support
- DPMS support

Text mode is supported even in higher resolutions, but there is limitation to lower pixclocks (maximum about 100 MHz). This limitation is not enforced by driver. Text mode supports 8bit wide fonts only (hardware limitation) and 16bit tall fonts (driver limitation).

There are two 4 bpp modes. First mode (selected if nonstd == 0) is mode with packed pixels, high nibble first. Second mode (selected if nonstd == 1) is mode with interleaved planes (1 byte interleave), MSB first. Both modes support 8bit wide fonts only (driver limitation).

Suspend/resume works on systems that initialize video card during resume and if device is active (for example used by fbcon).

37.3 Missing Features

(alias TODO list)

- secondary (not initialized by BIOS) device support
- MMIO support
- interlaced mode variant
- support for fontwidths != 8 in 4 bpp modes
- support for fontheight != 16 in text mode
- hardware cursor
- video overlay support
- vsync synchronization
- acceleration support (8514-like 2D, busmaster transfers)

37.4 Known bugs

- cursor disable in text mode doesn't work

-- Ondrej Zajicek <santiago@crfreenet.org>