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.

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.
- 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.
```

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.