OMAP2/3 Display Subsystem

This is an almost total rewrite of the OMAP FB driver in drivers/video/omap (let's call it DSS1). The main differences between DSS1 and DSS2 are DSI, TV-out and multiple display support, but there are lots of small improvements also.

The DSS2 driver (omapdss module) is in arch/arm/plat-omap/dss/, and the FB, panel and controller drivers are in drivers/video/omap2/. DSS1 and DSS2 live currently side by side, you can choose which one to use.

Features

Working and tested features include:

- MIPI DPI (parallel) output
- MIPI DSI output in command mode
- MIPI DBI (RFBI) output
- SDI output
- TV output
- All pieces can be compiled as a module or inside kernel
- Use DISPC to update any of the outputs
- Use CPU to update RFBI or DSI output
- OMAP DISPC planes
- RGB16, RGB24 packed, RGB24 unpacked
- YUV2, UYVY
- Scaling
- Adjusting DSS FCK to find a good pixel clock
- Use DSI DPLL to create DSS FCK

Tested boards include:
- OMAP3 SDP board
- Beagle board
- N810

omapdss driver

The DSS driver does not itself have any support for Linux framebuffer, V4L or such like the current ones, but it has an internal kernel API that upper level drivers can use.

The DSS driver models OMAP's overlays, overlay managers and displays in a flexible way to enable non-common multi-display configuration. In addition to modelling the hardware overlays, omapdss supports virtual overlays and overlay managers. These can be used when updating a display with CPU or system DMA.

Panel and controller drivers

The drivers implement panel or controller specific functionality and are not usually visible to users except through omapfb driver. They register themselves to the DSS driver.

omapfb driver

The omapfb driver implements arbitrary number of standard linux framebuffers. These framebuffers can be routed flexibly to any overlays, thus allowing very dynamic display architecture.

The driver exports some omapfb specific ioctls, which are compatible with the ioctls in the old driver.

The rest of the non standard features are exported via sysfs. Whether the final implementation will use sysfs, or ioctls, is still open.

V4L2 drivers

V4L2 is being implemented in TI.

From omapdss point of view the V4L2 drivers should be similar to framebuffer driver.

Architecture

Some clarification what the different components do:

- Framebuffer is a memory area inside OMAP's SRAM/SDRAM that contains the pixel data for the image. Framebuffer has width and height and color depth.
- Overlay defines where the pixels are read from and where they go on the screen. The overlay may be smaller than framebuffer, thus displaying only part of the framebuffer. The position of the overlay may be changed if the overlay is smaller than the display.
- Overlay manager combines the overlays in to one image and feeds them to display.
- Display is the actual physical display device.

A framebuffer can be connected to multiple overlays to show the same pixel data on all of the overlays. Note that in this case the overlay input sizes must be the same, but, in case of video overlays, the output size can be different. Any framebuffer can be connected to any overlay.

An overlay can be connected to one overlay manager. Also DISPC overlays can be connected only to DISPC overlay managers, and virtual overlays can be only connected to virtual overlays.

An overlay manager can be connected to one display. There are certain restrictions which kinds of displays an overlay manager can be connected:

- DISPC TV overlay manager can be only connected to TV display.
- Virtual overlay managers can only be connected to DBI or DSI displays.
- DISPC LCD overlay manager can be connected to all displays, except TV display.

Sysfs

The sysfs interface is mainly used for testing. I don't think sysfs
interface is the best for this in the final version, but I don't quite know
what would be the best interfaces for these things.

The sysfs interface is divided to two parts: DSS and FB.

```
/sys/class/graphics/fb? directory:
mirror          0=off, 1=on
rotate          Rotation 0-3 for 0, 90, 180, 270 degrees
rotate_type     0 = DMA rotation, 1 = VRFB rotation
overlays        List of overlay numbers to which framebuffer pixels go
phys_addr       Physical address of the framebuffer
virt_addr       Virtual address of the framebuffer
size            Size of the framebuffer

/sys/devices/platform/omapdss/overlay? directory:
enabled         0=off, 1=on
input_size      width,height (ie. the framebuffer size)
manager         Destination overlay manager name
name
output_size     width,height
position        x,y
screen_width    width
global_alpha    global alpha 0-255 0=transparent 255=opaque

/sys/devices/platform/omapdss/manager? directory:
display                         Destination display
name
alpha_blending_enabled          0=off, 1=on
trans_key_enabled               0=off, 1=on
trans_key_type                  gfx-destination, video-source
trans_key_value                 transparency color key (RGB24)
default_color                   default background color (RGB24)

/sys/devices/platform/omapdss/display? directory:
ctrl_name       Controller name
mirror          0=off, 1=on
update_mode     0=off, 1=auto, 2=manual
enabled         0=off, 1=on
name
rotate          Rotation 0-3 for 0, 90, 180, 270 degrees
timings         Display timings (pixclock,xres/hfp/hbp/hsw,yres/vfp/vbp/vsw)
                When writing, two special timings are accepted for tv-out:
                "pal" and "ntsc"
panel_name
tear_elim       Tearing elimination 0=off, 1=on
```

There are also some debugfs files at <debugfs>/omapdss/ which show information
about clocks and registers.

Examples

The following definitions have been made for the examples below:

```
ovl0=/sys/devices/platform/omapdss/overlay0
```

```
ovl1=/sys/devices/platform/omapdss/overlay1
ovl2=/sys/devices/platform/omapdss/overlay2

mgr0=/sys/devices/platform/omapdss/manager0
mgr1=/sys/devices/platform/omapdss/manager1

lcd=/sys/devices/platform/omapdss/display0
dvi=/sys/devices/platform/omapdss/display1
tv=/sys/devices/platform/omapdss/display2

fb0=/sys/class/graphics/fb0
fb1=/sys/class/graphics/fb1
fb2=/sys/class/graphics/fb2
```

Default setup on OMAP3 SDP

Here's the default setup on OMAP3 SDP board. All planes go to LCD. DVI and TV-out are not in use. The columns from left to right are: framebuffers, overlays, overlay managers, displays. Framebuffers are handled by omapfb, and the rest by the DSS.

```
FB0 --- GFX  -\            DVI
FB1 --- VID1 --+- LCD ---- LCD
FB2 --- VID2 -/   TV ----- TV
```

Example: Switch from LCD to DVI

```
w=`cat $dvi/timings | cut -d "," -f 2 | cut -d "/" -f 1`
h=`cat $dvi/timings | cut -d "," -f 3 | cut -d "/" -f 1`

echo "0" > $lcd/enabled
echo "" > $mgr0/display
fbset -fb /dev/fb0 -xres $w -yres $h -vxres $w -vyres $h
# at this point you have to switch the dvi/lcd dip-switch from the omap board
echo "dvi" > $mgr0/display
echo "1" > $dvi/enabled
```

After this the configuration looks like:

```
FB0 --- GFX  -\         -- DVI
FB1 --- VID1 --+- LCD -/   LCD
FB2 --- VID2 -/   TV ----- TV
```

Example: Clone GFX overlay to LCD and TV

```
w=`cat $tv/timings | cut -d "," -f 2 | cut -d "/" -f 1`
h=`cat $tv/timings | cut -d "," -f 3 | cut -d "/" -f 1`

echo "0" > $ovl0/enabled
echo "0" > $ovl1/enabled

echo "" > $fb1/overlays
echo "0,1" > $fb0/overlays
```

```
echo "$w,$h" > $ovl1/output_size
echo "tv" > $ovl1/manager

echo "1" > $ovl0/enabled
echo "1" > $ovl1/enabled

echo "1" > $tv/enabled
```

After this the configuration looks like (only relevant parts shown):

```
FB0 +-- GFX  ---- LCD ---- LCD
     \- VID1 ---- TV  ---- TV
```

Misc notes

OMAP FB allocates the framebuffer memory using the OMAP VRAM allocator.

Using DSI DPLL to generate pixel clock it is possible produce the pixel clock of 86.5MHz (max possible), and with that you get 1280x1024@57 output from DVI.

Rotation and mirroring currently only supports RGB565 and RGB8888 modes. VRFB does not support mirroring.

VRFB rotation requires much more memory than non-rotated framebuffer, so you probably need to increase your vram setting before using VRFB rotation. Also, many applications may not work with VRFB if they do not pay attention to all framebuffer parameters.

Kernel boot arguments

vram=<size>
- Amount of total VRAM to preallocate. For example, "10M". omapfb allocates memory for framebuffers from VRAM.

omapfb.mode=<display>:<mode>[,...]
- Default video mode for specified displays. For example, "dvi:800x400MR-24@60". See drivers/video/modedb.c. There are also two special modes: "pal" and "ntsc" that can be used to tv out.

omapfb.vram=<fbnum>:<size>[@<physaddr>][,...]
- VRAM allocated for a framebuffer. Normally omapfb allocates vram depending on the display size. With this you can manually allocate more or define the physical address of each framebuffer. For example, "1:4M" to allocate 4M for fb1.

omapfb.debug=<y|n>
- Enable debug printing. You have to have OMAPFB debug support enabled in kernel config.

omapfb.test=<y|n>
- Draw test pattern to framebuffer whenever framebuffer settings change. You need to have OMAPFB debug support enabled in kernel config.

```
omapfb.vrfb=<y|n>
        - Use VRFB rotation for all framebuffers.

omapfb.rotate=<angle>
        - Default rotation applied to all framebuffers.
          0 - 0 degree rotation
          1 - 90 degree rotation
          2 - 180 degree rotation
          3 - 270 degree rotation

omapfb.mirror=<y|n>
        - Default mirror for all framebuffers. Only works with DMA rotation.

omapdss.def_disp=<display>
        - Name of default display, to which all overlays will be connected.
          Common examples are "lcd" or "tv".

omapdss.debug=<y|n>
        - Enable debug printing. You have to have DSS debug support enabled in
          kernel config.

TODO
----

DSS locking

Error checking
- Lots of checks are missing or implemented just as BUG()

System DMA update for DSI
- Can be used for RGB16 and RGB24P modes. Probably not for RGB24U (how
  to skip the empty byte?)

OMAP1 support
- Not sure if needed
```