Product: Robinhood

CONFIDENTIAL

Not for use in any form without the explicit written permission from Beningo Engineering.

Approval List			
Prepared by:	Jacob Beningo		**Date:** 11-07-2013
Approved by:	Jacob Beningo		**Date:** 11-07-2013
Approved by:			**Date:**
Approved by:			**Date:**
Approved by:			**Date:**
Release Date:	11-07-2013	**Current Release Version: 1.0**	See Revision History

REVISION HISTORY:

Revision:	Description of Changes:	Date:	Revision By:
1.0	*Document Release*	*11.07.2013*	*JWB*

REFERENCES:

Revision:	Document:	Author:
1.0	*Modify Android to Support Different LCD Monitor*	*Runhong Deng*

Table of Contents

1. Situation Appraisal

Adding a new display to Android can either be a very simple endeavor, simply plug it in and it works, or it can be a complex and difficult task. There are a number of reasons for this that will be explained in this document. This document will help anyone interested in adding a new display to Android on the OMAP 5 platform.

NOTE: Some of the information contained herein is directly taken from the "Modify Android to Support Different LCD Monitors" document by Runhong Deng. This documents expands on the information and fills in missing pieces and lessons learned from integrating the Alpine displays for Robinhood.

2. Objectives

The objectives of this document is to:

- Explain how to get timing information for an LCD panel
- How to calculate the necessary driver parameters required to drive the display
- How to add the driver parameters to the Linux kernel
- Explain the differences in parameters between a HDMI and DPI driver configuration
- Provide some tips and tricks for debugging

3. LCD Timing Parameter (EDID)

Every display contains timing parameters that are stored inside of an EEPROM. This information is then sent using a standard protocol known as Extended Display Identification (EDID). This contains all of the information about the display timings such as refresh rate, resolution in the x and y axis's, pixel clock, margins and even the native and possible display standards.

There are a number of ways to get this information from the display including through the display datasheet. However, most likely this is not going to be available so another means is necessary. There are a number of software packages that can be downloaded for Windows and Linux that retrieve this information. They include:

Windows - **EDID Viewer utility -** http://www.eldim.fr/products/display-controller/fpdlite/fpdlite-free-tools

Linux - read-edid - http://polypux.org/projects/read-edid/

Download one these programs and run it to retrieve the EDID. The information retrieved will look like the example information below that was received from an Alpine Robinhood display. The complete information block can be seen in Figure 1. Despite all the information that is provided, the only real information of interest is under the timing characteristics; the Modeline.

Modeline..... "1920x1080" 138.700 1920 1968 2000 2080 1080 1083 1088 1111 +hsync –vsync

The modelne contains all of the information required that the display driver needs to properly setup a framebuffer with the LCD timing parameters. Below is an identification of what each parameter is

DCF	HR	SH1	SH2	HFL	VR	SV1	SV2	VFL
138.700	1920	1968	2000	2080	1080	1083	1088	1111

```
Monitor
  Windows description...... Generic PnP Monitor
  Manufacturer............. LG Display
  Plug and Play ID......... LGD040E
  Data string.............. 890N5€156WF4  [*CP437]
  Serial number............ n/a
  Manufacture date......... 2013, ISO week 0
  Filter driver............ None
  -------------------------
  EDID revision............ 1.4
  Input signal type........ Digital (DisplayPort)
  Color bit depth.......... 6 bits per primary color
  Color encoding formats... RGB 4:4:4, YCrCb 4:4:4
  Screen size.............. 350 x 190 mm (15.7 in)
  Power management......... Standby, Suspend, Active off/sleep
  Extension blocs.......... None
  -------------------------
  DDC/CI................... n/a

Color characteristics
  Default color space...... Non-sRGB
  Display gamma............ 2.20
  Red chromaticity......... Rx 0.635 - Ry 0.345
  Green chromaticity....... Gx 0.335 - Gy 0.623
  Blue chromaticity........ Bx 0.150 - By 0.052
  White point (default).... Wx 0.313 - Wy 0.329
  Additional descriptors... None

Timing characteristics
  Range limits............. Not available
  GTF standard............. Not supported
  Additional descriptors... None
  Preferred timing......... Yes
  Native/preferred timing.. 1920x1080p at 60Hz (16:9)
    Modeline............... "1920x1080" 138.700 1920 1968 2000 2080 1080 1083 1088 1111 +hsync -vsync

Standard timings supported

Report information
  Date generated........... 11/11/2013
  Software revision........ 2.70.0.989
  Data source.............. Registry-Active
  Operating system......... 6.1.7601.2.Service Pack 1

Raw data
  00,FF,FF,FF,FF,FF,FF,00,30,E4,0E,04,00,00,00,00,00,17,01,04,95,23,13,78,EA,9E,95,A2,58,55,9F,26,
  0D,50,54,00,00,00,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,2E,36,80,A0,70,38,1F,40,30,20,
  35,00,59,C2,10,00,00,1A,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,FE,00,38,
  39,30,4E,35,80,31,35,36,57,46,34,0A,00,00,00,00,00,00,41,31,9E,00,10,00,00,0A,01,0A,20,20,00,AC
```

Figure 1 – EDID Information

4. Calculating Frame Buffer Parameters

Unfortunately the EDID information is not directly used by the frame buffer. Instead, this information is used to search an array of video display settings which are then used by the frame buffer to drive the display. This means that the EDID information need to be manually converted into a form that can be entered into the "database" of valid display timings.

The display timings are stored in modedb.c inside of cea_modes[] but this will be discussed later. First the EDID information needs to be converted. There is information useful for this that can be found at *https://www.kernel.org/doc/Documentation/fb/framebuffer.txt* . The easiest way to see what to do is to follow the following calculation procedure that uses the above EDID information as an example.

Start by calculating the pixel clock in terms of picoseconds rather than MHz.

DCF	HR	SH1	SH2	HFL	VR	SV1	SV2	VFL
138.700	1920	1968	2000	2080	1080	1083	1088	1111

1) Calculate the pixel clock

   Pixclock = 1,000,000 / DCF
   Pixelclock = 1,000,000 / 138.500 = **7220**

2) Calculate the Horizontal Timings

   Left_margin = HFL – SH2
   Left_margin = 2080 – 2000 = **80**

   Right_margin = SH1 – HR
   Right_margin = 1968 – 1920 = **48**

   Hsync_len = SH2 – SH1
   Hsync_len = 2000 – 1968 = **32**

3) Calculate Vertical Timings

   Upper_margin = VFL – SV2
   Upper_margin = 1111 – 1088 = **23**

   Lower_margin = SV1 – VR
   Lower_margin = 1083 – 1080 = **3**

   Vsync_len = SV2 – SV1
   Vsync_len = 1088 – 1083 = **5**

5. Adding a HDMI Display

Unfortunately the EDID information isn't used to calculate the driver parameters but instead are used to select the timing parameters. If a match is not found then the default video mode is selected which is something on the order of 640x480. In order to ensure that the EDID is able to be matched, the timing parameters for the display need to be added to the cea_modes array.

The cea_modes array can be found in the file modedb.c. It is located under /kernel/android-3.4/drivers/video/ . An example of how the array structure looks by default can be found below (keep in mind that there would be 64 entries):

```
const struct fb_videomode cea_modes[CEA_MODEDB_SIZE] = {
    {},
    /* 1: 640x480p @ 59.94Hz/60Hz */
    {.refresh = 59, .xres = 640, .yres = 480, .pixclock = 39721,
     .left_margin = 48, .right_margin = 16,
     .upper_margin = 33, .lower_margin = 10,
     .hsync_len = 96, .vsync_len = 2,
     .sync = 0,
     .flag = FB_FLAG_RATIO_4_3,
     .vmode = FB_VMODE_NONINTERLACED},
…….};
```

Now the values that were calculated earlier for the driver can be substituted in for the last entry or more appropriately the size of the cea_modes array can be increased by one and then the new timings entered into the array. Below is an example of how the Alpine display settings are entered in:

```
    /* 65: 1920x1080p @ 60Hz */
    {.refresh = 60, .xres = 1920, .yres = 1080, .pixclock = 7210,
     .left_margin = 80, .right_margin = 48,
     .upper_margin = 23, .lower_margin = 3,
     .hsync_len = 32, .vsync_len = 5,
     .sync = FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT,
     .flag = FB_FLAG_RATIO_16_9,
     .vmode = FB_VMODE_NONINTERLACED},
};
```

At this point saving the file and then recompiling the image are necessary in order for the new settings to be included in the kernel.

6. Adding a DPI Display

The Robinhood binary image supports multiple monitors provided that the monitors have the same resolution. The two displays are driven through HDMI and DPI. The previous section showed how to add a new HDMI display. Adding a DPI display is not much different. The EDID information is still needed although the information is added in a different location. The DPI setting can be found in the generic_dpi_panels[] array located in panel-generic-dpi.c which is in /kernel/android-3.4/drivers/video/omap2/displays/.

The DPI port by default uses the lp101_panel settings. These settings can be commented out and in their place the new display timings can be added. An example for the Alpine displays can be found below:

```
/* Panel configurations */
static struct panel_config generic_dpi_panels[] = {

    /* lp101_panel */
    {

        {
            /* 1280 x 800 @ 60 Hz  Reduced blanking VESA CVT 0.31M3-R */
            .x_res          = 1920,
            .y_res          = 1080,
            .pixel_clock    = 148500,
            .hfp            = 32,
            .hsw            = 48,
            .hbp            = 80,
            .vfp            = 5,
            .vsw            = 3,
            .vbp            = 23,
        },
        .acbi                   = 0x0,
        .acb                    = 0x0,
        .config                 = OMAP_DSS_LCD_TFT,
        .power_on_delay                 = 0,
        .power_off_delay        = 0,
        .name                   = "generic_lp101",
    },
```

Note that the pixel clock is specified in MHz instead of picoseconds! This is different than modedb.c!

7. Troubleshooting

7.1 Forcing the Video Mode

Despite adding the timing information into the kernel, sometimes the display subsystem is just not able to match the EDID information to a display timing. In this case the default resolution will be used even when the valid timings are available. Thankfully there is a method for forcing the mode which will guarantee that the video settings you want are actually used.

This is done in hdmi.c which can be found by doing a search of the kernel folder. It will most likely be located in /kernel/android-3.4/drivers/video/omap2/dss/. There is a function called set_timings which loops through the cea_modes array and tries to match the EDID information to the timing settings. Instead of allowing the software to perform this check, the cea_modes index can be set to the desired value, the mode set and then the function exited, forcing the video mode. Below is an example implementation that can be used to achieve this where the changes are highlighted in red:

```
static int hdmi_set_timings(struct fb_videomode *vm, bool check_only)
{
	int i = 0;
	int r = 0;
	DSSDBG("hdmi_set_timings\n");

	i = 64;

	hdmi.ip_data.cfg.cm.code = i;

	hdmi.ip_data.cfg.cm.mode = HDMI_HDMI;
	hdmi.ip_data.cfg.timings =
	cea_modes[hdmi.ip_data.cfg.cm.code];
	goto done;

	if (!vm->xres || !vm->yres || !vm->pixclock)
		goto fail;

	for (i = 0; i < CEA_MODEDB_SIZE; i++) {
		if (relaxed_fb_mode_is_equal(cea_modes + i, vm)) {
			*vm = cea_modes[i];
			if (check_only)
				return 1;
			hdmi.ip_data.cfg.cm.code = i;

			hdmi.ip_data.cfg.cm.mode = HDMI_HDMI;
			hdmi.ip_data.cfg.timings =
			cea_modes[hdmi.ip_data.cfg.cm.code];
			goto done;
		}
	}
.
.
.
.
.
}
```

7.2 Verify that the pixel clock is not outside of range

There are some ranges of pixel clock that the internal PLL is unable to reach. The result is that an error occurs and the display is not driven. Make sure to check the datasheet that the range is supported.

7.3 DPI is in MHz

Unfortunately the DPI and HDMI pixel clocks are specified in different units! Make sure that you are using MHz for DPI and picoseconds for HDMI!

7.4 Check the actual display mode during runtime

The display mode can be checked during runtime. That is the display timings can be reported out so that you can compare if the mode the driver is in is what you are expecting. This information can be accesed from the android console using the ADB shell. An example for the DPI display would be as follows:

cat /sys/devices/omapdss/display0/timings

the result is something similar to

153600, 1920/48/80/32,1080/3/23/5

An example of the HDMI would be as follows:

cat /sys/devices/omapdss/display1/timings

the result is something similar to

138696, 1920/48/80/32,1080/3/23/5

Note that the two display timings should match and the pixel clocks should be similar.

7.5 Displays must be the same resolution

Despite the ability to specify different panel resolutions, the 5AJ.1.5.1 binary cannot support monitors with different resolutions. They default to the resolution of the DPI display which is the primary display in dual monitor mode. The HDMI display (DISPLAY1) will report it is in the mode you are forcing but the driver will override that mode without telling you!