AM335X_LCD 分析

先从板级文件看起，board-am335xevm.c(/arch/arm/mach-omap2/)文件中定义了板级平台描述结构：

```
MACHINE_START(AM335XEVM, "am335xevm")
	/* Maintainer: Texas Instruments */
	.atag_offset	= 0x100,
	.map_io		= am335x_evm_map_io,
	.init_early	= am33xx_init_early,
	.init_irq	= ti81xx_init_irq,
	.handle_irq	= omap3_intc_handle_irq,
	.timer		= &omap3_am33xx_timer,
	.init_machine	= am335x_evm_init,
MACHINE_END
```

系统初始化，通过机器码(/include/generated/mach-types.h 中定义)找到对应的这个平台描述结构，后续会执行此结构中指定的函数。

此处与 LCD 相关的有：am33xx_init_early() 及 am335x_evm_init()

1.先看 am335x_evm_init()函数

```
static void __init am335x_evm_init(void)
{
	……
	/*main setup*/
	setup_ok335x();
	……
}

/* ok335x */
static void setup_ok335x(void)
{
	……
	_configure_device(EVM_SK, ok335x_dev_cfg, PROFILE_NONE);
	……
};

/* ok335x*/
static struct evm_dev_cfg ok335x_dev_cfg[] = {
	……
	{lcd_init , DEV_ON_BASEBOARD, PROFILE_ALL},
	……
	{NULL, 0, 0},
};

static void lcd_init(int evm_id, int profile)
```

```
{
	setup_pin_mux(lcdc_pin_mux);// 引脚功能配置

	if (conf_disp_pll(300000000)) {
		pr_info("Failed configure display PLL, not attempting to"
			"register LCDC\n");
		return;
	}// 时钟配置
	TFC_S9700RTWV35TR_01B_pdata.get_context_loss_count =
omap_pm_get_dev_context_loss_count;
	if (am33xx_register_lcdc(&TFC_S9700RTWV35TR_01B_pdata))
		pr_info("Failed to register LCDC device\n");// 根据设置好的属性进行设备注册

	return;
}// 这个函数我做过改动
```

以下三个结构定义了 lcd 的一些参数，与具体驱动程序中定义的 lcd 的参数共同起作用。移植的时候需要根据屏幕的参数修改此处的参数设置及驱动程序中的参数设置，后边会提到。

```
/* djf 20150105 add start */
struct da8xx_lcdc_platform_data TFC_S9700RTWV35TR_01B_pdata = {
	.manu_name		= "ThreeFive",
	.controller_data	= &lcd_cfg,
	.type			= "AT070TN94V_1",
};
/* djf 20150105 add end */

#if 1
//djf 20150105 add
static const struct display_panel disp_panel = {
	QVGA,
	32,
	32,
	COLOR_ACTIVE,
};
static struct lcd_ctrl_config lcd_cfg = {
	&disp_panel,
	.ac_bias		= 255,
	.ac_bias_intrpt		= 0,
	.dma_burst_sz		= 16,
	.bpp			= 32,
	.fdd			= 255,
	.tft_alt_mode		= 0,
	.stn_565_mode		= 0,
	.mono_8bit_mode		= 0,
```

```
	.invert_line_clock	= 1,
	.invert_frm_clock	= 1,
	.sync_edge		= 0,
	.sync_ctrl		= 1,
	.raster_order		= 0,
	.fifo_th		= 6,
};
#endif
```

参数设置好，接下来分析下 am33xx_register_lcdc() 这个函数：
位置在/arch/arm/mach-omap2/devices.c 文件中。

```
int __init am33xx_register_lcdc(struct da8xx_lcdc_platform_data *pdata)
{
	int id = 0;
	struct platform_device *pdev;
	struct omap_hwmod *oh;
	char *oh_name = "lcdc";
	char *dev_name = "da8xx_lcdc";

	oh = omap_hwmod_lookup(oh_name);
	if (!oh) {
		pr_err("Could not look up LCD%d hwmod\n", id);
		return -ENODEV;
	}

	pdev = omap_device_build(dev_name, id, oh, pdata,
			sizeof(struct da8xx_lcdc_platform_data), NULL, 0, 0);
	if (IS_ERR(pdev)) {
		WARN(1, "Can't build omap_device for %s:%s.\n",
			dev_name, oh->name);
		return PTR_ERR(pdev);
	}
	return 0;
}
```

<1>此处主要调用两个函数，第一个 omap_hwmod_lookup()

```
/**
 * omap_hwmod_lookup - look up a registered omap_hwmod by name
 * @name: name of the omap_hwmod to look up
 *
 * Given a @name of an omap_hwmod, return a pointer to the registered
 * struct omap_hwmod *, or NULL upon error.
 */
struct omap_hwmod *omap_hwmod_lookup(const char *name)
```

```
{
	struct omap_hwmod *oh;

	if (!name)
		return NULL;

	oh = _lookup(name);

	return oh;
}

/**
 * _lookup - find an omap_hwmod by name
 * @name: find an omap_hwmod by name
 *
 * Return a pointer to an omap_hwmod by name, or NULL if not found.
 */
static struct omap_hwmod *_lookup(const char *name)
{
	struct omap_hwmod *oh, *temp_oh;

	oh = NULL;

	list_for_each_entry(temp_oh, &omap_hwmod_list, node) {
		if (!strcmp(name, temp_oh->name)) {
			oh = temp_oh;
			break;
		}
	}

	return oh;
}
```

此处发现遍历了一个链表：omap_hwmod_list ，那么这个链表是在哪里建立好的，即在哪里将各节点添加进链表的？这就要看上面提到的第二个函数了（此处与 LCD 相关的有：am33xx_init_early() 及 am335x_evm_init() ）。

2. /arch/arm/mach-omap2/io.c 中函数：

```
void __init am33xx_init_early(void)
{
	……
	am33xx_hwmod_init();
	……
```

```
}

static __initdata struct omap_hwmod *am33xx_hwmods[] = {
	……
	/* LCDC class */
	&am33xx_lcdc_hwmod,
	……
};

int __init am33xx_hwmod_init(void)
{
	return omap_hwmod_register(am33xx_hwmods);
}
```

注：以下部分是 am33xx_lcdc_hwmod 相关信息，这里定义了 lcd 的 resource 资源：

```
/* lcdc */
static struct omap_hwmod_class_sysconfig lcdc_sysc = {
	.rev_offs  = 0x0,
	.sysc_offs = 0x54,
	.sysc_flags	= (SYSC_HAS_SIDLEMODE | SYSC_HAS_MIDLEMODE),
	.idlemodes	= (SIDLE_FORCE | SIDLE_NO | SIDLE_SMART),
	.sysc_fields	= &omap_hwmod_sysc_type2,
};

static struct omap_hwmod_class am33xx_lcdc_hwmod_class = {
	.name		= "lcdc",
	.sysc	= &lcdc_sysc,
};

static struct omap_hwmod_irq_info am33xx_lcdc_irqs[] = {
	{ .irq = 36 },
	{ .irq = -1 }
};

struct omap_hwmod_addr_space am33xx_lcdc_addr_space[] = {
	{
		.pa_start  = 0x4830E000,
		.pa_end		= 0x4830E000 + SZ_8K - 1,
		.flags		= ADDR_MAP_ON_INIT | ADDR_TYPE_RT,
	},
	{ }
};

struct omap_hwmod_ocp_if am33xx_l3_main__lcdc = {
```

```
	.master		= &am33xx_l3_main_hwmod,
	.slave		= &am33xx_lcdc_hwmod,
	.addr		= am33xx_lcdc_addr_space,
	.user		= OCP_USER_MPU,
};

static struct omap_hwmod_ocp_if *am33xx_lcdc_slaves[] = {
	&am33xx_l3_main__lcdc,
};

static struct omap_hwmod am33xx_lcdc_hwmod = {
	.name		= "lcdc",
	.class		= &am33xx_lcdc_hwmod_class,
	.clkdm_name	= "lcdc_clkdm",
	.mpu_irqs	= am33xx_lcdc_irqs,
	.main_clk= "lcdc_fck",
	.prcm		= {
		.omap4	= {
			.clkctrl_offs	= AM33XX_CM_PER_LCDC_CLKCTRL_OFFSET,
			.modulemode = MODULEMODE_SWCTRL,
		},
	},
	.slaves		= am33xx_lc	dc_slaves,
	.slaves_cnt	= ARRAY_SIZE(am33xx_lcdc_slaves),
	.flags		= (HWMOD_SWSUP_SIDLE | HWMOD_SWSUP_MSTANDBY),
};
```

再看下：return omap_hwmod_register() 这个函数
/arch/arm/mach-omap2/omap-hwmod.c 文件中：

```
/**
 * omap_hwmod_register - register an array of hwmods
 * @ohs: pointer to an array of omap_hwmods to register
 *
 * Intended to be called early in boot before the clock framework is
 * initialized.	If @ohs is not null, will register all omap_hwmods
 * listed in @ohs that are valid for this chip.	Returns 0.
 */
int __init omap_hwmod_register(struct omap_hwmod **ohs)
{
	int r, i;

	if (!ohs)
```

```
		return 0;

	i = 0;
	do {
		r = _register(ohs[i]);
		WARN(r, "omap_hwmod: %s: _register returned %d\n", ohs[i]->name,
			r);
	} while (ohs[++i]);

	return 0;
}

/**
 * _register - register a struct omap_hwmod
 * @oh: struct omap_hwmod *
 *
 * Registers the omap_hwmod @oh.	Returns -EEXIST if an omap_hwmod
 * already has been registered by the same name; -EINVAL if the
 * omap_hwmod is in the wrong state, if @oh is NULL, if the
 * omap_hwmod's class field is NULL; if the omap_hwmod is missing a
 * name, or if the omap_hwmod's class is missing a name; or 0 upon
 * success.
 *
 * XXX The data should be copied into bootmem, so the original data
 * should be marked __initdata and freed after init.	This would allow
 * unneeded omap_hwmods to be freed on multi-OMAP configurations.	Note
 * that the copy process would be relatively complex due to the large number
 * of substructures.
 */
static int __init _register(struct omap_hwmod *oh)
{
	int ms_id;

	if (!oh || !oh->name || !oh->class || !oh->class->name ||
		(oh->_state != _HWMOD_STATE_UNKNOWN))
		return -EINVAL;

	pr_debug("omap_hwmod: %s: registering\n", oh->name);

	if (_lookup(oh->name))
		return -EEXIST;

	ms_id = _find_mpu_port_index(oh);
	if (!IS_ERR_VALUE(ms_id))
```

```
        oh->_mpu_port_index = ms_id;
    else
        oh->_int_flags |= _HWMOD_NO_MPU_PORT;

    list_add_tail(&oh->node, &omap_hwmod_list);

    spin_lock_init(&oh->_lock);

    oh->_state = _HWMOD_STATE_REGISTERED;

    /*
     * XXX Rather than doing a strcmp(), this should test a flag
     * set in the hwmod data, inserted by the autogenerator code.
     */
    if (!strcmp(oh->name, MPU_INITIATOR_NAME))
        mpu_oh = oh;

    return 0;
}
```

好的，到这里终于找到向链表 omap_hwmod_list 中添加节点了。既然链表中已经添加好了，那么再回到 1.<1>，此处主要调用两个函数，第一个 omap_hwmod_lookup() ，分析下第二个函数：omap_device_build() /arch/arm/plat-omap/omap_device.c 中：

```
/**
 * omap_device_build - build and register an omap_device with one omap_hwmod
 * @pdev_name: name of the platform_device driver to use
 * @pdev_id: this platform_device's connection ID
 * @oh: ptr to the single omap_hwmod that backs this omap_device
 * @pdata: platform_data ptr to associate with the platform_device
 * @pdata_len: amount of memory pointed to by @pdata
 * @pm_lats: pointer to a omap_device_pm_latency array for this device
 * @pm_lats_cnt: ARRAY_SIZE() of @pm_lats
 * @is_early_device: should the device be registered as an early device or not
 *
 * Convenience function for building and registering a single
 * omap_device record, which in turn builds and registers a
 * platform_device record.    See omap_device_build_ss() for more
 * information.    Returns ERR_PTR(-EINVAL) if @oh is NULL; otherwise,
 * passes along the return value of omap_device_build_ss().
 */
struct platform_device *omap_device_build(const char *pdev_name, int pdev_id,
                                          struct omap_hwmod *oh, void *pdata,
                                          int pdata_len,
```

```
                                struct omap_device_pm_latency *pm_lats,
                                int pm_lats_cnt, int is_early_device)
{
        struct omap_hwmod *ohs[] = { oh };

        if (!oh)
                return ERR_PTR(-EINVAL);

        return omap_device_build_ss(pdev_name, pdev_id, ohs, 1, pdata,
                                    pdata_len, pm_lats, pm_lats_cnt,
                                    is_early_device);
}

/**
 * omap_device_build_ss - build and register an omap_device with multiple hwmods
 * @pdev_name: name of the platform_device driver to use
 * @pdev_id: this platform_device's connection ID
 * @oh: ptr to the single omap_hwmod that backs this omap_device
 * @pdata: platform_data ptr to associate with the platform_device
 * @pdata_len: amount of memory pointed to by @pdata
 * @pm_lats: pointer to a omap_device_pm_latency array for this device
 * @pm_lats_cnt: ARRAY_SIZE() of @pm_lats
 * @is_early_device: should the device be registered as an early device or not
 *
 * Convenience function for building and registering an omap_device
 * subsystem record.    Subsystem records consist of multiple
 * omap_hwmods.    This function in turn builds and registers a
 * platform_device record.    Returns an ERR_PTR() on error, or passes
 * along the return value of omap_device_register().
 */
struct platform_device *omap_device_build_ss(const char *pdev_name, int pdev_id,
                                struct omap_hwmod **ohs, int oh_cnt,
                                void *pdata, int pdata_len,
                                struct omap_device_pm_latency *pm_lats,
                                int pm_lats_cnt, int is_early_device)
{
        int ret = -ENOMEM;
        struct platform_device *pdev;
        struct omap_device *od;

        if (!ohs || oh_cnt == 0 || !pdev_name)
                return ERR_PTR(-EINVAL);

        if (!pdata && pdata_len > 0)
```

```
        return ERR_PTR(-EINVAL);

    pdev = platform_device_alloc(pdev_name, pdev_id);//  内存分配
    if (!pdev) {
        ret = -ENOMEM;
        goto odbs_exit;
    }

    /* Set the dev_name early to allow dev_xxx in omap_device_alloc */
    if (pdev->id != -1) // id 为 0
        dev_set_name(&pdev->dev, "%s.%d", pdev->name,    pdev->id);//da8xx_ lcdc0
    else
        dev_set_name(&pdev->dev, "%s", pdev->name);

    od = omap_device_alloc(pdev, ohs, oh_cnt, pm_lats, pm_lats_cnt); //  这里的一部分工作
是设置 platform 的 resource，而具体的 resource 是在上面提到的 am33xx_lcdc_hwmod  中定
义的
    if (!od)
        goto odbs_exit1;

    ret = platform_device_add_data(pdev, pdata, pdata_len);//  这个 pdata 是板级文件中定义
的 TFC_S9700RTWV35TR_01B_pdata
    if (ret)
        goto odbs_exit2;

    if (is_early_device) // is_early_device  值为 0
        ret = omap_early_device_register(pdev);
    else
        ret = omap_device_register(pdev);
    if (ret)
        goto odbs_exit2;

    return pdev;

odbs_exit2:
    omap_device_delete(od);
odbs_exit1:
    platform_device_put(pdev);
odbs_exit:

    pr_err("omap_device: %s: build failed (%d)\n", pdev_name, ret);

    return ERR_PTR(ret);
}
```

```
/**
 * omap_device_register - register an omap_device with one omap_hwmod
 * @od: struct omap_device * to register
 *
 * Register the omap_device structure.    This currently just calls
 * platform_device_register() on the underlying platform_device.
 * Returns the return value of platform_device_register().
 */
int omap_device_register(struct platform_device *pdev)
{
        pr_debug("omap_device: %s: registering\n", pdev->name);

        pdev->dev.parent = &omap_device_parent;
        pdev->dev.pm_domain = &omap_device_pm_domain;
        return platform_device_add(pdev);
}
```

分析到这里，总算是看到 lcd 设备被注册成了 platform 设备了。到此，lcd 的设备注册就算完毕了。