**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。