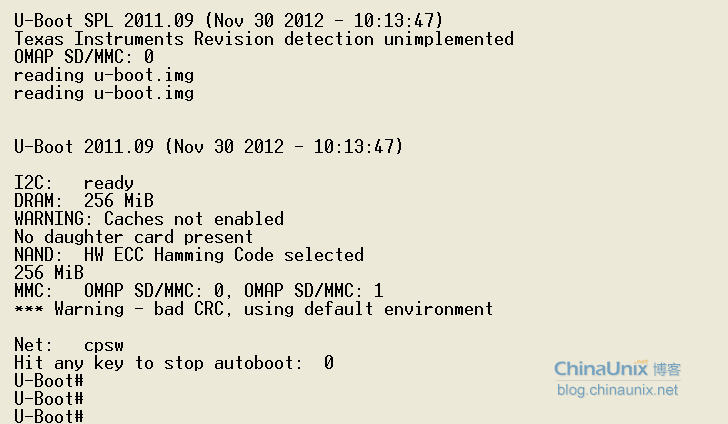
Uboot2011 代码分析：

正确uboot输出

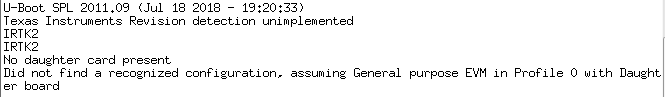


Mmc\_initialize(bd\_t \*bis)[Mmc.c中]🡪Board\_mmc\_init🡪Omap\_mmc\_init(int dev\_index) “OMAP SD/MMC: 0”

Jump\_to\_image\_no\_args🡪Spl\_mmc\_load\_image🡪Mmc\_load\_image\_fat[omap-common]🡪File\_fat\_read🡪“reading u-boot.img”

Init\_func\_i2c🡪 “I2C: ready”

自己输出



Step1:Cpu\_init\_crit[start.s]🡪Lowlevel\_init.S🡪s\_init(void) 🡪void preloader\_console\_init(void) “Texas Instruments …..”

Step2:board\_init\_f 🡪relocate\_code(CONFIG\_SPL\_STACK, &gdata, CONFIG\_SPL\_TEXT\_BASE);

其中gdata 🡪 static gd\_t gdata \_\_attribute\_\_ ((section(".data")));

Step3:在RAM中执行：

Board\_init\_r[start.s]🡪board\_init(void) 🡪void detect\_daughter\_board(void) “No daughter card present”

board\_init(void) 🡪 “Did not find a recognized configuration,”

board\_init

->configure\_evm\_pin\_mux

->static struct evm\_pin\_mux \*am335x\_evm\_pin\_mux[] (配置引脚)

cpu\_init\_crit

|🡪lowlevel\_init

| |🡪s\_init

| | |🡪l2\_cache\_enable(); //可以不初始化

| | |🡪Disable WDT //关闭看门狗

| | |🡪pll\_init(); //PLL和时钟设置

| | | |🡪mpu\_pll\_config(MPUPLL\_M\_500); //mpu时钟初始化

| | | |🡪per\_clocks\_enable //初始化外围模块时钟,MMC0,MMC1

| | |🡪rtc32k\_enable(); //使能RTC

| | |🡪enable\_uart0\_pin\_mux(); //串口引脚等配置

| | |🡪init\_timer(); //初始化定时器

| | |🡪preloader\_console\_init(); //初始化串口

| | |🡪i2c\_init //初始化IIC

| | |🡪if (read\_eeprom()) //读取EEPROM板子信息，**需修改代码，修改的代**| | | //**码在附录中**

| | | **可剪裁:**enable\_gpio0\_7\_pin\_mux(); //EVM SK 1.2A及更高版本使用gpio0\_7

| | | //启用DDR3。

| | | gpio\_request(GPIO\_DDR\_VTT\_EN, "ddr\_vtt\_en");

| | | gpio\_direction\_output(GPIO\_DDR\_VTT\_EN, 1);

| | |🡪config\_am335x\_ddrx //初始化DDR2,3

| | | |🡪enable\_ddr3\_clocks(); //初始化DDR3时钟

| | | |🡪set IO control registers //设置IO控制寄存器

| | | | writel(DDR3\_IOCTRL\_VALUE, DDR\_CMD0\_IOCTRL);

| | | | writel(DDR3\_IOCTRL\_VALUE, DDR\_CMD1\_IOCTRL);

| | | |🡪config\_emif\_ddr3(); //不知道干啥的

**SPL编译时使用的board\_init\_r**

board\_init\_r

|🡪mem\_malloc\_init

|🡪timer\_init();

|🡪i2c\_init(CONFIG\_SYS\_I2C\_SPEED, CONFIG\_SYS\_I2C\_SLAVE);

|🡪void **spl\_board\_init**(void)[evm.c]

| |🡪board\_init(); //板载初始化，初始化复用引脚

| | |🡪enable\_i2c0\_pin\_mux(); //初始化IIC

| | |🡪read\_eeprom()

| | | /\***读取开发板EEPROM，移植需修改，根据读到的信息， 判断板卡型号**

| | | **GP\_BOARD、IA\_BOARD、IPP\_BOARD（板卡有两部分组成）**

| | |  **BONE\_BOARD、SK\_BOARD （板卡单独一个整体 ）\*/**

|||🡪detect\_daughter\_board(); //判断是否有子板

| | |🡪static struct am335x\_baseboard\_id \_\_attribute\_\_((section (".data"))) **header**;

| | |/\*根据**header.config**和**header.name**配置 board\_id,profile,

| | |daughter\_board\_connected给下面的configure\_evm\_pin\_mux初始化复用引脚

| | |🡪**configure\_evm\_pin\_mux**

| | | 🡪set\_evm\_pin\_mux(am335x\_evm\_pin\_mux[dghtr\_brd\_id], profile,

| | | daughter\_board\_flag);

| | | //其中am335x\_evm\_pin\_mux是初始化引脚的具体，修改引脚可在这里

| | |🡪gpmc\_init(); //用于初始化**SDRAM和SRAM**，**IRTK2有问题需修改，** | | | **公司使用的DDR3是GPMC接口？**

| |🡪PMIC //电源管理，使用到了IIC0，**这里有问题，需检查**

| | //**IRTK2的PMIC管理芯片与EVM基本一致**

| | //**PMIC\_INT1\_GPIO:** EVM接GPMC\_CLK/IRTK2接GPMC\_CSn0

| | //

| |🡪mpu\_pll\_config(MPUPLL\_M\_720); //**这里有问题，需检查**

|🡪boot\_device = omap\_boot\_device();

| 🡪return (u32) (boot\_params.omap\_bootdevice);

|🡪switch (boot\_device) { //选择加载image的方式，有MMC,NAND,SPI等

| |🡪 #ifdef CONFIG\_SPL\_MMC\_SUPPORT //如果定义了MMC支持

| | |🡪spl\_mmc\_load\_image(); //加载image

| | |🡪mmc\_initialize(); //mmc初始化**,经过测试这里卡住**

| | |🡪find\_mmc\_device(0); //寻找MMC，**可能需修改0**

| | |🡪err = mmc\_init(mmc); //mmc初始化

| | |🡪boot\_mode = omap\_boot\_mode();

| | | |🡪u32 omap\_bootmode = MMCSD\_MODE\_FAT; // MMCSD\_MODE\_FAT==2;

| | | |🡪mmc\_load\_image\_fat(mmc); //加载fat格式

| | | | |🡪struct image\_header \*header;

| | | | | header = (struct image\_header \*)(**CONFIG\_SYS\_TEXT\_BASE** -

| | | | | **sizeof(struct image\_header)**);

| | | | | //其中 #define **CONFIG\_SYS\_TEXT\_BASE** **0x80100000**

|🡪switch (spl\_image.os)

| |🡪jump\_to\_image\_no\_args(); **//uboot的入口地址**

| | |🡪image\_entry\_noargs\_t image\_entry =

| | | (image\_entry\_noargs\_t) **spl\_image.entry\_point**;

| | | **//跳转到entry\_point执行，这个u-boot.img头部信息提供。**

**enable\_IRTK2\_mmc1\_pin\_mux(); //init irtk2 mmc1 pin**

**Uboot编译时使用的board\_init\_r**

UBOOT 环境参数 CONFIG\_EXTRA\_ENV\_SETTINGS

Uboot 中使用

原来为：mmc\_root=/dev/mmcblk0p2 rw

mmc\_root\_fs\_type=ext3 rootwait

修改为：mmc\_root=/dev/ram rw

mmc\_root\_fs\_type=cramfs

mach-types.h

**gd\_t ：global\_data 数据结构的定义，位于：/arch/arm/include/asm/global\_data.h**

**bd\_t ：bd\_info 数据结构的定义，位于：/arch/arm/include/asm/u-boot.h 中。**

**#define DECLARE\_GLOBAL\_DATA\_PTR register volatile gd\_t \*gd asm ("r8")**

**DECLARE\_GLOBAL\_DATA\_PTR 宏定义在系统初始化过程中会被频繁调用**

其的作用是，声明gd这么一个全局的指针，这个指针指向gd\_t结构体类型，并且这个gd指针是保存在ARM的r8这个寄存器里面的。

gb\_t 结构体中某些元素的值是来自于 uboot.img's header，这个header的数据保存在内存的0x807FFFCO，大小为 64字节

**Uboot编译时使用的board\_init\_r**

|🡪board\_init\_r [arch/arm/lib/board.c]

| |🡪enable\_caches();

| |🡪board\_init();

| |🡪#ifdef CONFIG\_SERIAL\_MULTI

| | serial\_initialize(); //未进入

| |🡪mem\_malloc\_init

| |🡪#if !defined(CONFIG\_SYS\_NO\_FLASH)

| | flash\_init();

| |🡪#if defined(CONFIG\_CMD\_NAND)

| | nand\_init();

| |🡪#ifdef CONFIG\_GENERIC\_MMC

| | mmc\_initialize(bd);

| |🡪env\_relocate(); //初始化环境变量

| |🡪stdio\_init(); //获取设备列表

| |🡪jumptable\_init(); //gd->jt

| |🡪console\_init\_r(); //控制台完全初始化

| |🡪interrupt\_init(); //异常中断初始化

| |🡪enable\_interrupts(); //使能异常中断

| |🡪eth\_initialize(gd->bd);    //初始化以太网 /net/eth.c文件中

| |🡪main\_loop(); //进入uboot主循环

| | |🡪#ifndef CONFIG\_SYS\_HUSH\_PARSER

| | | static char lastcommand[CONFIG\_SYS\_CBSIZE] = { 0, }; //定义一个字符数组来存

| | | //储所有读取到的命令

| | |🡪u\_boot\_hush\_start

| | |🡪hush\_init\_var

| | |🡪run\_command

| | |🡪len = readline (CONFIG\_SYS\_PROMPT); //读取串口信息

Uboot命令处理过程分析：

命令的结构以及定义：

#define U\_BOOT\_CMD(name,maxargs,rep,cmd,usage,help) \

U\_BOOT\_CMD\_COMPLETE(name,maxargs,rep,cmd,usage,help,NULL)

#define U\_BOOT\_CMD\_COMPLETE(name,maxargs,rep,cmd,usage,help,comp) \

cmd\_tbl\_t \_\_u\_boot\_cmd\_##name Struct\_Section = \

U\_BOOT\_CMD\_MKENT\_COMPLETE(name,maxargs,rep,cmd,usage,help,comp)

#define U\_BOOT\_CMD\_MKENT(name,maxargs,rep,cmd,usage,help) \

U\_BOOT\_CMD\_MKENT\_COMPLETE(name,maxargs,rep,cmd,usage,help,NULL)

#define U\_BOOT\_CMD\_MKENT\_COMPLETE(name,maxargs,rep,cmd,usage,help,comp) \

{#name, maxargs, rep, cmd, usage, \_CMD\_HELP(help) \_CMD\_COMPLETE(comp)}

根据以上定义可以分析出：

**#define** U\_BOOT\_CMD(name,maxargs,rep,cmd,usage,help) \

**cmd\_tbl\_t** \_\_u\_boot\_cmd\_##name **Struct\_Section** =\

{#name, maxargs, rep, cmd, usage, \_CMD\_HELP(help) \_CMD\_COMPLETE(comp)}

1.**cmd\_tbl\_t**是命令格式的数据结构，其定义如下所示、

|  |
| --- |
| struct cmd\_tbl\_s {  char \*name; /\* Command Name \*/  int maxargs; /\* maximum number of arguments \*/  int repeatable; /\* autorepeat allowed? \*/  /\* Implementation function \*/  int (\*cmd)(struct cmd\_tbl\_s \*, int, int, char \* const []);  char \*usage; /\* Usage message (short) \*/  #ifdef CONFIG\_SYS\_LONGHELP  char \*help; /\* Help message (long) \*/  #endif  #ifdef CONFIG\_AUTO\_COMPLETE  /\* do auto completion on the arguments \*/  int (\*complete)(int argc, char \* const argv[], char last\_char, int maxv, char \*cmdv[]);  #endif  }; |

2. ##name是指直接用U\_BOOT\_CMD里面的name参数替换，#name表示将U\_BOOT\_CMD里面的name参数加上双引号（“”），然后替换

3. **Struct\_Section**是个宏定义，定义在common/command.h

#define struct\_section \_\_attribute\_\_((unused,section(".u\_boot\_cmd")))

它表示将该处的内容放入u\_boot\_cmd段（具体u\_boot\_cmd段的信息可以参看u-boot.lds连接文件）

|  |
| --- |
| 例如，定义了如下命令 U\_BOOT\_CMD(command,1,0,fun,”short help”,”long help”)  cmd\_tbl\_t \_\_u\_boot\_cmd\_command \  \_\_attribute\_\_((unused.section(".u\_boot\_cmd")))={"command",1,0,fun,"short help‘,"long help"’} |

二、处理控制台输入命令

UBOOT 环境参数 CONFIG\_EXTRA\_ENV\_SETTINGS

Uboot主要命令在 cmd\_nvedit.c中

在s5pc210\_ubivers.h中定义：

#define CONFIG\_ENV\_IS\_IN\_MMC 1

#define CONFIG\_SYS\_MMC\_ENV\_DEV 0

#define CONFIG\_ENV\_SIZE 4096

#define CONFIG\_ENV\_OFFSET ((32 - 4) << 10)/\* 32KiB - 4KiB \*/

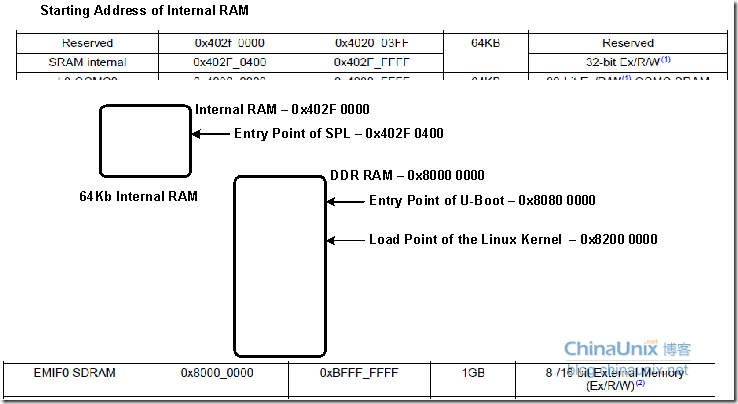
#define CONFIG\_ENV\_OFFSET (6 \* 64 \* 1024)

#define CONFIG\_ENV\_SIZE (8 \* 1024)

#define CONFIG\_ENV\_IS\_IN\_MMC

#define CONFIG\_SYS\_MMC\_ENV\_DEV 0

（BOOTLOADER）AM335x bootloader 分3部分： 1st stage：RBL （ROM） 2st stage：SPL （内部RAM：0x402f0400） 3st stage：U-Boot （一般外部DDR） 注：SPL的代码包含在U-Boot中，在编译U-Boot时会自动编译。



声卡：

|🡪evm\_hw\_params

| |🡪machine\_is\_am335xevm() set sysclk

| |🡪snd\_soc\_dai\_set\_fmt(codec\_dai, AUDIO\_FORMAT); //entry wm8960\_set\_dai\_fmt

| |🡪 snd\_soc\_dai\_set\_fmt(cpu\_dai, AUDIO\_FORMAT); //entry davinci\_mcasp\_set\_dai\_fmt

| |🡪snd\_soc\_dai\_set\_sysclk(codec\_dai, 0, sysclk, SND\_SOC\_CLOCK\_OUT); **//error**

soc\_pcm\_hw\_params[soc-pcm.c]

static struct snd\_soc\_dai\_ops wm8960\_dai\_ops = {

.hw\_params = wm8960\_hw\_params,

.digital\_mute = wm8960\_mute,

.set\_fmt = wm8960\_set\_dai\_fmt,

**.set\_clkdiv = wm8960\_set\_dai\_clkdiv,**

**.set\_pll = wm8960\_set\_dai\_pll,**

};

static struct snd\_soc\_dai\_ops wm8962\_dai\_ops = {

.hw\_params = wm8962\_hw\_params,

.set\_sysclk = wm8962\_set\_dai\_sysclk,

.set\_fmt = wm8962\_set\_dai\_fmt,

.digital\_mute = wm8962\_mute,

};

static struct snd\_soc\_dai\_ops aic3x\_dai\_ops = {

.hw\_params = aic3x\_hw\_params,

.digital\_mute = aic3x\_mute,

.set\_sysclk = aic3x\_set\_dai\_sysclk,

.set\_fmt = aic3x\_set\_dai\_fmt,

};

static struct snd\_soc\_dai\_driver wm8960\_dai = { [wm8960.c]

static struct snd\_soc\_dai\_driver davinci\_mcasp\_dai[] [davinci-mcasp.c]

输出信息：

[ 7.781463] soc\_pcm\_hw\_params

[ 7.784393] snd\_soc\_dai\_set\_fmt

[ 7.797546] snd\_soc\_dai\_set\_fmt

[ 7.801116] entry davinci\_mcasp\_set\_dai\_fmt

[ 7.805572] entry dai->driver && dai->driver->ops->set\_sysclk

[ 7.805572]

[ 7.813568] entry davinci\_mcasp\_hw\_params

[ 7.818054] exit davinci\_mcasp\_hw\_params



由以上代码可以得知：

**配置时定义了NAND支持**

/\* NAND support \*/

#ifdef CONFIG\_NAND

**#define BOOT\_DEVICE\_MMC1 8**

**#define BOOT\_DEVICE\_MMC2 9 /\* eMMC or daughter card \*/**

**/\*mmc引脚配置\*/**

static struct module\_pin\_mux mmc1\_pin\_mux[] = {

{OFFSET(gpmc\_ad3), (MODE(1) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_DAT3 \*/

{OFFSET(gpmc\_ad2), (MODE(1) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_DAT2 \*/

{OFFSET(gpmc\_ad1), (MODE(1) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_DAT1 \*/

{OFFSET(gpmc\_ad0), (MODE(1) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_DAT0 \*/

{OFFSET(gpmc\_csn1), (MODE(2) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_CLK \*/

{OFFSET(gpmc\_csn2), (MODE(2) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_CMD \*/

{OFFSET(gpmc\_csn0), (MODE(7) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_WP \*/

{OFFSET(gpmc\_advn\_ale), (MODE(7) | RXACTIVE | PULLUP\_EN)}, /\* MMC1\_CD \*/

{-1},

};

**Strncmp 若str1与str2的前n个字符相同，则返回0**

#ifndef \_\_ASSEMBLY\_\_

struct omap\_boot\_parameters {

char \*boot\_message;

unsigned int mem\_boot\_descriptor;

unsigned char omap\_bootdevice;

unsigned char reset\_reason;

unsigned char ch\_flags;

};

#endif

/\*

\* This is used to verify if the configuration header

\* was executed by rom code prior to control of transfer

\* to the bootloader. SPL is responsible for saving and

\* passing the boot\_params pointer to the u-boot.

\*/

struct omap\_boot\_parameters **boot\_params** \_\_attribute\_\_ ((section(".data")));

/\*

\* store the boot params passed from rom code or saved

\* and passed by SPL

\*/

cmp r0, #0

beq 1f

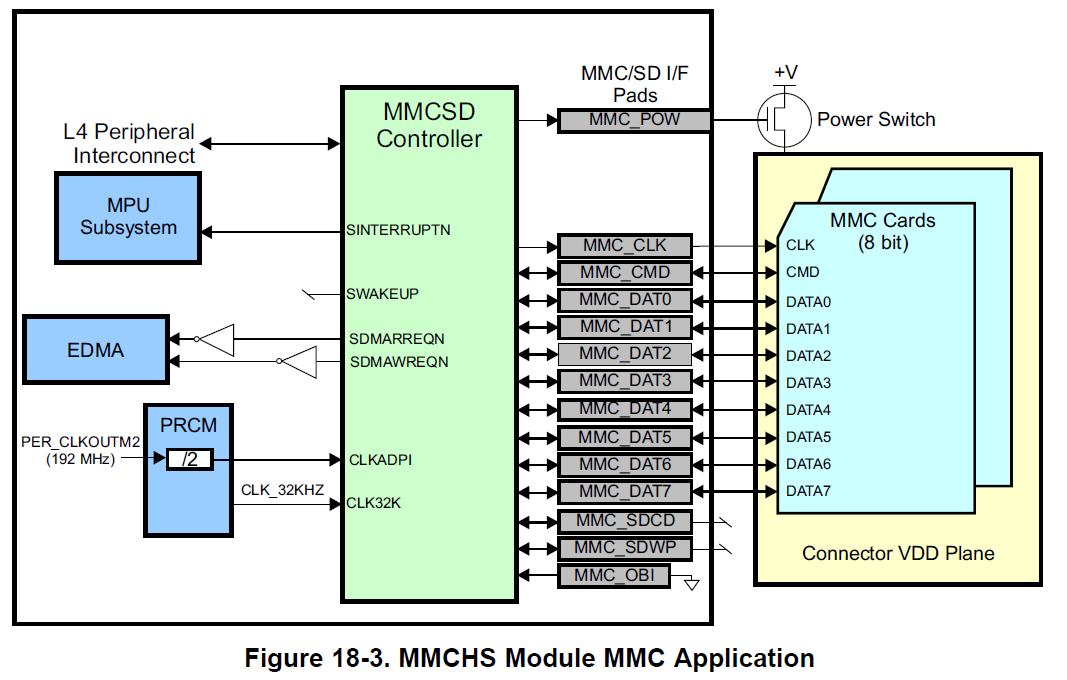
ldr r1, =**boot\_params**

str r0, [r1]

emmc中启动：



Emmc 为8bit连接方式



WiFi配置：

UART 串口代码分析笔记：

Serial.c --->配置DMA

am335x\_evm\_init

omap\_serial\_init

|🡪Omap\_serial\_init(void) //入口函数

| |🡪omap\_serial\_board\_init(NULL) //轮询所有uart,并配置基本属性

| | |🡪list\_for\_each\_entry(uart, &uart\_list, node) // uart\_list怎么来的？

| | | |🡪omap\_serial\_init\_port

| | | | |🡪omap\_serial\_init\_port(&bdata, NULL); //用于初始化串口 \*info->NULL

| | | | |   /\*配置所有属性，DMD等

\*不要混合调用omap\_serial\_init\_port（）和omap\_serial\_init（），

  \*只使用两者中的一个。\*/

| | | | |🡪

Mmc代码分析： 需要分为spl 和uboot两段？？？

|🡪Mmc\_initialize //mmc初始化，uboot段

| |🡪 board\_mmc\_init(bis)

| | |🡪omap\_mmc\_init[omap\_hsmmc.c] //uboot初始化时使用

| | | | mmc->send\_cmd = mmc\_send\_cmd[omap\_hsmmc.c] //发送命令函数

| | | | mmc->set\_ios = mmc\_set\_ios; //

| | | | **mmc->init = mmc\_init\_setup;**

| |

|🡪mmc\_init\_setup[omap\_hsmmc.c] // struct mmc类型的结构体指针, 记录了mmc的

| //一些信息，以及函数指针所指向的函数

| hsmmc\_t \*mmc\_base = (hsmmc\_t \*)**mmc->priv**; //mmc->priv =

| // (hsmmc\_t \*)OMAP\_HSMMC1\_BASE; 0x 48060100

| // (hsmmc\_t \*)OMAP\_HSMMC2\_BASE; 0x 481D8100

| |🡪mmc\_board\_init[omap\_hsmmc.c] //好像没有做初始化？

| |🡪

**增加mmc调试信息：#ifdef CONFIG\_MMC\_TRACE**

**设置sysclkd寄存器的**

**当前uboot中使用save保存环境变量，是写调用nand的写入，需修改为mmc的写入函数。**

mmc\_startup ---> mmc\_set\_bus\_width(mmc, 8);

emmc调试：

uboot不配置mmc0,mmc1不配置mux情况一样？

|🡪mmc\_init

| |🡪mmc->has\_init //判断mmc是否已经初始化

| |🡪mmc->init(mmc) // 也就是进入了**mmc\_init\_setup**[omap\_hsmmc.c]

| |🡪mmc\_set\_bus\_width(mmc, 1); // 设置mmc数据宽度

| | mmc->bus\_width = width;

| | mmc\_set\_ios(mmc); // mmc\_set\_ios

| |🡪mmc\_set\_clock(mmc, 1); // 设置时钟,该函数设置时钟在

| | //omap\_mmc\_init中设置的最大时钟和最小时钟相关,mmc1可能有问题？？？？

| |🡪mmc\_go\_idle(mmc);

| |🡪 err = mmc\_send\_if\_cond(mmc); //测试是否是SD\_VERSION\_2，

| | //mmc0为SD\_VERSION\_2

| | //**mmc1不是SD\_VERSION\_2的卡**

| |🡪err = sd\_send\_op\_cond(mmc); //尝试获取SD卡的运行状况

| | |🡪mmc\_send\_cmd(mmc, &cmd, NULL); //发送到cmd55命令，检测是否正确

| |🡪if (err == TIMEOUT) {

| | |🡪mmc\_send\_op\_cond (mmc); //如果不是SD卡运行条件，使用mmc卡检测

| | | |🡪mmc\_send\_cmd(mmc, &cmd, NULL); //发送cmd1命令，emmc在这里发送

| | | | //**整机错误**

| | | |

| | if (err)

| | printf("Card did not respond to voltage select!\n");} // **mmc1输出该信息，修改**

| |🡪mmc\_startup //mmc 初始化

| | |🡪mmc\_send\_cmd[MMC\_CMD\_ALL\_SEND\_CID] //发送**CMD2**,获取设备**CID**号

| | |🡪mmc\_send\_cmd[SD\_CMD\_SEND\_RELATIVE\_ADDR] //发送CMD3,设置相对地址，

| | | //进入Standby状态

| | |🡪 mmc\_send\_cmd[MMC\_CMD\_SEND\_CSD] //发送**cmd9**,

| | |🡪mmc\_send\_status(mmc, timeout); //等待准备状态

| | | |🡪MMC\_CMD\_SEND\_STATUS //发送**CMD13**

| | |🡪 freq = fbase[(cmd.response[0] & 0x7)]; //设置频率

| | | mult = multipliers[((cmd.response[0] >> 3) & 0xf)];

| | | mmc->tran\_speed = freq \* mult;

| | | mmc->read\_bl\_len = 1 << ((cmd.response[1] >> 16) & 0xf);

| | | mmc->write\_bl\_len = 1 << ((cmd.response[3] >> 22) & 0xf);

| | |🡪if (!IS\_SD(mmc) && (mmc->version >= MMC\_VERSION\_4)) //进入该函数

| | | |🡪 mmc\_send\_ext\_csd //发送CMD8,

| | |🡪 mmc->card\_caps |= MMC\_MODE\_4BIT; //变为4线？

| | |🡪 mmc\_change\_freq //mmc改变频率

| | | |🡪mmc\_send\_ext\_csd

其中CMD8

Cmd\_mmc.c 🡪"mmc%d(part %d) is current device\n"

在擦写emmc时发生错误，如下所示：

|  |
| --- |
| U-Boot# mmc erase 1 1  MMC erase: dev # 1, block # 1, count 1 ... CMD\_SEND:35  ARG 0x00000001  FLAG 0  mmc\_send\_cmd: timedout waiting for cmddis!  MMC\_RSP\_R1,5,6,7 0x00000000  mmc erase failed  0 blocks erase: ERROR |

公司输出：

公司打印的输出为：mmc1(part 0) is current device

|  |
| --- |
| U-Boot# mmc erase 1 1  MMC erase: dev # 1, block # 1, count 1 ... 1 blocks erase: OK |

Mmcinfo 公司输出

|  |
| --- |
| SD卡  Device: OMAP SD/MMC  Manufacturer ID: 9c  OEM: 534f  Name: USD00  Tran Speed: 25000000  Rd Block Len: 512  SD version 2.0  High Capacity: Yes  Capacity: 7.4 GiB  Bus Width: 4-bit  emmc:  Device: OMAP SD/MMC  Manufacturer ID: 11  OEM: 100  Name: 008G7  Tran Speed: 25000000  Rd Block Len: 512  MMC version 4.0  High Capacity: Yes  Capacity: 7.3 GiB  Bus Width: 4-bit |

自己调试输出

|  |
| --- |
| SD卡  Device: OMAP SD/MMC  Manufacturer ID: 9c  OEM: 534f  Name: USD00  Tran Speed: 25000000  Rd Block Len: 512  SD version 2.0  High Capacity: Yes  Capacity: 7.4 GiB  Bus Width: 4-bit  emmc:  Device: OMAP SD/MMC  Manufacturer ID: 3c  OEM: 3c00  Name: Tran Speed: 0  Rd Block Len: 1  MMC version 1.2  High Capacity: Yes  Capacity: 1 KiB  Bus Width: 8-bit |

|🡪mmc\_send\_cmd

| |🡪printf("%s: timedout waiting for cmddis!\n", \_\_func\_\_);

mmc命令分析

|🡪do\_mmcops

| |🡪if (strcmp(argv[1], "rescan") == 0) //如果输入的是重新扫描

| | |🡪mmc\_init(mmc) //重新初始化

| |🡪if (strncmp(argv[1], "part", 4) == 0) //如果是part0部分

| | |🡪mmc\_init(mmc);

| | |🡪mmc\_dev = mmc\_get\_dev(curr\_device);

| | |🡪print\_part(mmc\_dev);

**移植uboot2011笔记**

**SPL代码修改：**

**Step1:**

cpu\_init\_crit[start.S]

|🡪lowlevel\_init

| |🡪s\_init

| | |🡪if (read\_eeprom())

**int read\_eeprom(void)**

|  |
| --- |
| int read\_eeprom(void)  {  **#if 1**  **strcpy(header.name,"IRTK2");**  **//printf("IRTK2\n");**  **#else**  /\* Check if baseboard eeprom is available \*/  if (i2c\_probe(I2C\_BASE\_BOARD\_ADDR)) {  printf("Could not probe the EEPROM; something fundamentally "  "wrong on the I2C bus.\n");  return 1;  }  /\* read the eeprom using i2c \*/  if (i2c\_read(I2C\_BASE\_BOARD\_ADDR, 0, 2, (uchar \*)&header,  sizeof(header))) {  printf("Could not read the EEPROM; something fundamentally"  " wrong on the I2C bus.\n");  return 1;  }  if (header.magic != 0xEE3355AA) {  /\* read the eeprom using i2c again, but use only a 1 byte address \*/  if (i2c\_read(I2C\_BASE\_BOARD\_ADDR, 0, 1, (uchar \*)&header,  sizeof(header))) {  printf("Could not read the EEPROM; something fundamentally"  " wrong on the I2C bus.\n");  return 1;  }  if (header.magic != 0xEE3355AA) {  printf("Incorrect magic number in EEPROM\n");  return 1;  }  }  **#endif**  return 0;  } |

**Step2:**

if (!strncmp("A335X\_SK", header.name, 8) || !strncmp("IRTK2", header.name, 5)) {

is\_ddr3 = 1;

/\*

\* EVM SK 1.2A and later use gpio0\_7 to enable DDR3.

\* This is safe enough to do on older revs.

\*/

enable\_gpio0\_7\_pin\_mux();

gpio\_request(GPIO\_DDR\_VTT\_EN, "ddr\_vtt\_en");

gpio\_direction\_output(GPIO\_DDR\_VTT\_EN, 1);

}

if (!strncmp("SKU#01", header.config, 6)) {

board\_id = GP\_BOARD;

detect\_daughter\_board\_profile();

} else if (!strncmp("SKU#02", header.config, 6)) {

board\_id = IA\_BOARD;

detect\_daughter\_board\_profile();

} else if (!strncmp("SKU#03", header.config, 6)) {

board\_id = IPP\_BOARD;

} else if (!strncmp("A335BONE", header.name, 8)) {

board\_id = BONE\_BOARD;

profile = 1; /\* profile 0 is internally considered as 1 \*/

daughter\_board\_connected = 0;

} else if (!strncmp("A335X\_SK", header.name, 8) **|| !strncmp("IRTK2", header.name, 5)** ) {

board\_id = SK\_BOARD;

profile = 1; /\* profile 0 is internally considered as 1 \*/

daughter\_board\_connected = 0;

} else {

printf("Did not find a recognized configuration, "

"assuming General purpose EVM in Profile 0 with "

"Daughter board\n");

board\_id = GP\_BOARD;

profile = 1; /\* profile 0 is internally considered as 1 \*/

daughter\_board\_connected = 1;

}

Am335x\_evm.h

#define PHYS\_DRAM\_1\_SIZE 0x8000000 /\* 128 MiB \*/

支持uboot环境变量save到mmc中命令：

|  |
| --- |
| 在am335x\_evm中删除SPI和 NAND相关支持  修改  #ifndef CONFIG\_SPI\_BOOT  **#define CONFIG\_MMC\_ENV**  #endif  添加以下代码:  #if defined(**CONFIG\_MMC\_ENV**)  #undef CONFIG\_ENV\_IS\_NOWHERE  #define CONFIG\_ENV\_IS\_IN\_MMC 1  #define CONFIG\_SYS\_MMC\_ENV\_DEV 0  #define CONFIG\_ENV\_SIZE 4096  #define CONFIG\_ENV\_OFFSET ((32 - 4) << 10)/\* 32KiB - 4KiB \*/  #endif/\* MMC support \*/ |

修改uboot环境变量，需注意：一旦在uboot中使用save命令，那么之后uboot都是启动save在内存中的环境变量，**修改uboot源码的环境变量就不会被读取，源码具体待分析。**

|  |
| --- |
| #define CONFIG\_BOOTCOMMAND \  "mmc dev ${mmc\_dev};" \  "if mmc rescan; then " \  "if run mmc\_load\_uimage; then "\  "run mmc\_args;"\  "echo \*\*\*booting from extern storage\*\*\*;"\  "bootm ${kloadaddr};" \  "fi;"\  "fi;"\  "mmc dev ${mmc\_regular};"\  "if mmc rescan; then "\  "if run emmc\_load\_uimage; then "\  "run mmc\_args;"\  "bootm ${kloadaddr};"\  "fi;"\  "fi;"\  展开：setenv bootargs root=/dev/ram rw initrd=0x81600000,32MB rootfstype=cramfs ip=none console=ttyO0,115200  set bootargs noinitrd root=/dev/nfs nfsroot=192.168.10.169:/work/rootfs/fs\_mini ip=192.168.10.17:192.168.10.169:192.168.10.1:255.255.255.0::usb0:off init=/linuxrc console=ttyO0 |

公司的uboot：bootcmd

bootcmd=mmc dev ${mmc\_dev}; if mmc rescan; then if run mmc\_load\_uimage; then run mmc\_args;echo \*\*\*booting from extern storage\*\*\*; bootm ${kloadaddr};fi;fi;mmc dev ${mmc\_regular};if mmc rescan; then if run emmc\_load\_uimage; then run mmc\_args;bootm ${kloadaddr};fi;fi;

xld\_size\_blks=0xc8

xld\_start\_blk=0x100

kstart\_blk=0x800

k\_size\_blks=0x1400

fs\_size\_blks=0x1666

fs\_start\_blk=0x2800

ub\_buff=0x81060000

ub\_size\_blks=0x200

ub\_start\_blk=0x300

/\*擦除emmc全部内容\*/

erase\_mmc\_all=mmc dev ${mmc\_regular}; echo \*\*\* saving partiton table \*\*\*;mmc read ${ptable\_buff} 0 1;echo erasing bootloader...; mmc erase ${xld\_start\_blk} 800; echo erasing kernel...; mmc erase ${kstart\_blk} ${k\_size\_blks}; echo erasing rootfs...; mmc erase ${fs\_start\_blk} ${fs\_size\_blks}

/\*读取全部内容到内存中\*/

mmc\_fat\_read\_all=mmc dev ${mmc\_update};fatload mmc ${mmc\_update} ${spl\_buff} ${mlo};fatload mmc ${mmc\_update} ${ub\_buff} ${u-boot}; fatload mmc ${mmc\_update} ${knl\_buff} ${bootfile}; fatload mmc ${mmc\_update} ${fs\_buff} ${ramdisk}

/\*写入内容到emmc中\*/

update\_all=mmc dev ${mmc\_regular};echo flashing bootloader...;mmc write ${spl\_buff} ${xld\_start\_blk} ${xld\_size\_blks};mmc write ${ub\_buff} ${ub\_start\_blk} ${ub\_size\_blks};echo flashing kernel...;mmc write ${knl\_buff} ${kstart\_blk} ${k\_size\_blks}; echo flashing rootfs...;mmc write ${fs\_buff} ${fs\_start\_blk} ${fs\_size\_blks};echo \*\*\* recovering partition table \*\*\*;mmc write ${ptable\_buff} 0 1;

/\*更新系统，先擦除全部，在读取到内存中，最后写入\*/

update\_sys=echo entering update mode...;mmc dev ${mmc\_update}; if mmc rescan; then echo found SD/MMC in slot(1), enter upgrade mode.;if run erase\_mmc\_all; then run mmc\_fat\_read\_all;run update\_all;echo update system sucessfully!;echo Please reboot and unplug the card...;run halt;fi;fi;

内核代码分析

板载初始化定义[board-am335xevm.c]

static struct omap2\_hsmmc\_info am335x\_mmc[] \_\_initdata = {

在最上层的板级初始化文件（board-am335xevm.c）中会定义一个这样的结构体数组，确定所要初始化的引脚复用寄存器，交由omap\_mux\_init\_signals(partition, board\_mux);使用。例如：

/\*内核初始化\*/

static struct omap\_board\_mux board\_mux[] \_\_initdata = {

修改mmc1引脚：

static void mmc1\_init(int evm\_id, int profile)