

Read The F*ing Source Code

不忘初心

CnBlogs Home New Post Contact Admin Rss Posts - 9 Articles - 0 Comments - 0

Search

ZZK

Google

My Tags

Linux sdio mmc(1)
系统调用 中断处理 异同(1)

Post Categories

general IO driver(2)
lisp(1)
OS(5)
sisp
软件漫话(1)

Post Archives

2014/4 (1)
2014/3 (1)
2014/1 (1)
2013/12 (3)
2013/4 (2)
2011/10 (1)

Top Posts

- 1. Linux内核之mmc子系统-sdio(4243)
- 2. 系统调用和中断处理的异同（以Linux MIPS为例）(1015)
- 3. Linux内核中SPI/I2c子系统剖析(771)
- 4. Linux kernel驱动相关抽象概念及其实现 之“bus, device, driver”(586)
- 5. eCos下针对MIPS指令集的backtrace(559)

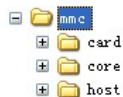
Linux内核之mmc子系统-sdio

现在的Linux内核中，mmc不仅是一个驱动，而是一个子系统。这里通过分析Linux3.2.0内核，结合TI的arm335x平台及omap_hsmmcd host分析下mmc子系统，重点关注sdio及架构在其上的具体sdio IP驱动实现。

1. General overview

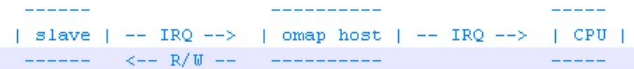
1.1 源码概览

Linux kernel把mmc,sd以及sdio三者的驱动代码整合在一起，俗称mmc子系统。源码位于drivers/mmc下。其下有三个子目录，分别是：



其中，card用于构建一个块设备作为上层与mmc子系统沟通的桥梁；core抽象了mmc,sd,sdio三者的通用操作；host则是各类平台上的host驱动代码，包括如TI Omap的omap_hsmmc,三星的s3cmci等。

1.2 硬件层IP对象间的联系



即，cpu要访问slave必须通过host进行，包括slave的中断。omap host的具体组成及其与slave之间的连线如图1.1所示：

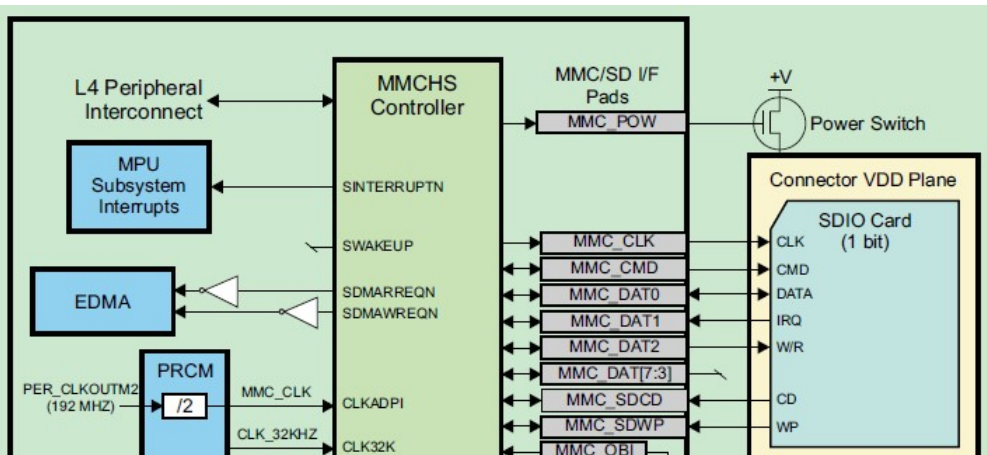




图1.1

和别的中断控制器一样，host的MPU中断子系统有一个仲裁机制，分别使用IE和ISE控制是否向CPU报中断以及是否care slave报出来的中断。

Host在数据传输时支持8bit模式，但slave是sdio时仅支持1-bit & 4-bit模式，当支持4-bit模式时，data1和IRQ线复用。

注意：MMCHS的smart-idle wake up line(SWAKEUP)没有连出去，在别的一些host中，它是和PRCM子系统连接，用于在MMCHS处于suspend时自动提醒PRCM给MMCHS提供clock。

2. 从bus,driver,device看mmc子系统

Linux下的任何驱动在内核中最终都抽象为bus, driver以及device三者间的相互作用。

2.1 mmc下的bus,driver,device模型

Mmc子系统涉及到三条总线。

Host驱动相应的driver和device挂载在Linux内核内置的虚拟抽象总线platform_bus_type。两者的匹配采用名称匹配的方式，即driver和device两者的name一样则认为该device对应该driver，这里是“omap_hsmmc”。

Card驱动相应的driver和device挂载在mmc自己创建的虚拟总线mmc_bus_type下，直接匹配。

Sdio驱动相应的driver和device挂载在mmc自己创建的虚拟总线sdio_bus_type下，ID匹配。

注意：Linux内核中，匹配函数默认使用bus注册的匹配函数，如果bus没有注册则使用driver注册的匹配函数。所以，一般自己创建虚拟总线时，其匹配函数和driver的匹配函数都是一致的。

2.2 按时间顺序观察mmc中各bus, driver, device对象初始化流程

2.2.1 Host device对象

Host device对象首先被初始化并挂载到platform_bus_type，但是这个过程不在mmc子系统源码下，它在平台初始化过程中init了，具体的流程参考5.1。

2.2.2 mmc_bus,sdio_bus对象

core初始化时，件core.c中的subsys_initcall(mmc_init)创建这两条mmc自己的虚拟总线。

2.2.3 card driver对象

Card初始化时，文件block.c中module_init(mmc_blk_init);函数创建mmcblk driver对象，并将之挂载到mmc_bus_type总线上。

2.2.4 host driver对象

Host初始化时，文件omap_hsmmc.c中module_init(omap_hsmmc_init);函数创建“omap_hsmmc”driver对象。

2.2.5 card device对象

到2.2.4为止，host的device和driver都已经创建，匹配后调用host对应的probe函数：omap_hsmmc_probe()。该函数将进行检测slave、初始化slave等操作。

检测sdio并创建card device对象。换言之，这个device对象并没有对应的硬件设备，它只是抽象出来用于和他人交互的一个虚拟device。具体的流程参考5.1。

2.2.6 sdio device对象

紧接着2.2.5后面，在函数sdio_init_func()函数中将创建一个device对象并挂载到sdio_bus_type上。具体的流程参考5.1。

2.2.7 小结

到此为止，mmc下三条总线、六个device/driver对象只差一个sdio driver了。这个driver对象对应着具体的sdio IP驱动，比如sdio_wifi, sdio_uart等。这样来看sdio IP驱动其实是构建在mmc子系统之上的。

3. 上层与mmc以及mmc内部模块之间的交互方式

从软件层面看mmc内部模块交互以及外部访问mmc的方式，如图3.1所示，。

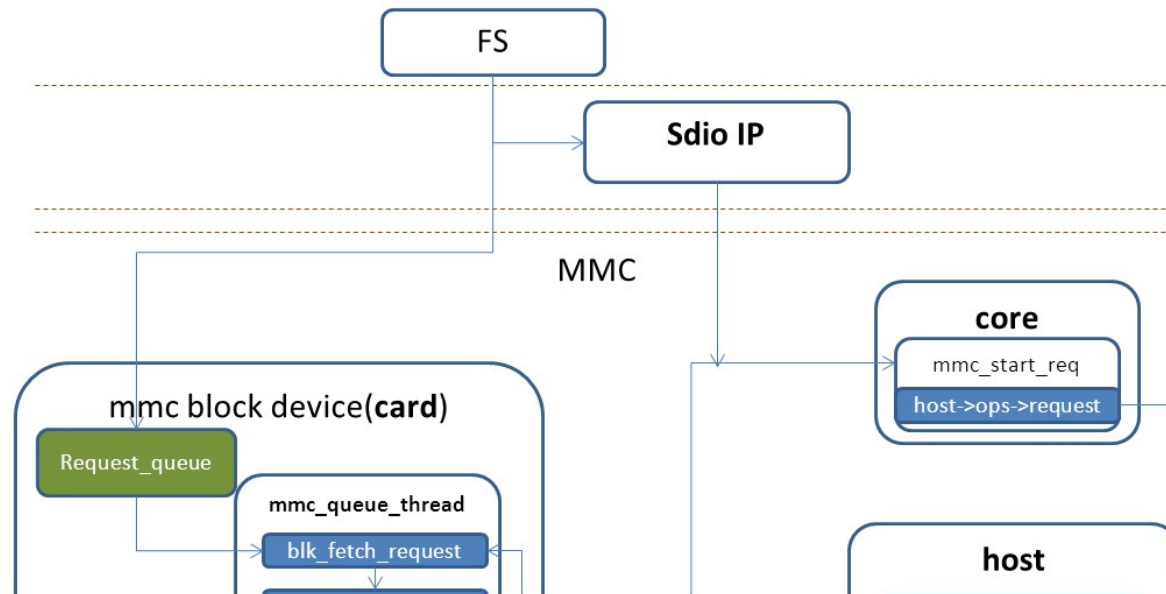


图3.1

如之前所言，mmc内部的core模块是各类host、sd及sdio操作的抽象，访问host的通用行为都放在其中，而各类host将其各自独特的行为注册到host规定的接口上。Mmc提供了两种方式供外部访问，mmc子系统本身对外呈现的是一个块设备，应用层通过VFS到具体的FS再到块设备逐层访问，块设备通过core让host和slave进行通信，sd卡的访问就是这种典型方式。另外一种方式是不通过块设备访问core，而是在驱动层新建一个驱动，这个驱动构建在mmc之上，让它有权限访问core，具有sdio接口的IP驱动就可以这么做。至于这个sdio的IP驱动对外以何种方式呈现由它自己决定，抽象为一个块设备或者字符设备都可以。

需要注意的是：core内部访问host的操作有睡眠动作，一般的访问行为是这样的：

- 1) 获取host资源（mmc_claim_host），可能sleep；
- 2) 发送访问请求（如读/写）；
- 3) 等待（block）本次访问结果（无论slave是否response，host都会通过中断给出返回值）；
- 4) 释放host资源（mmc_release_host）。

4. sdio

4.1 sdio和sd/mmc的差异

sdio事件通过card interrupt通知host，在host已有的中断处理过程中进行，如图4.1所示。

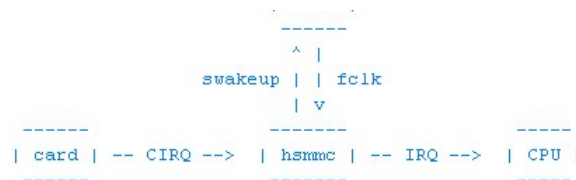


图4.1

但是sdio的中断事件（以CIRQ bit位表示）的处理和别的事件有点差异。Sdio中断事件的处理可能需要sleep。

比如读操作，它的流程是：1）来一个CIRQ表明sdio中的FIFO已经达到阈值；2）host告诉上层这个事件；3）sdio中断处理例程将去sdio FIFO中读取数据，它是通过host的标准操作进行的，如之前所言，这个过程会阻塞。而同样的读操作对sd卡则不需要，它的流程是：1）host发出读操作；2）slave把准备好的数据放到host的FIFO中并assert中断；3）host处理自己的FIFO并通知上层读操作结束。

所以对于sdio而言，host只是充当sdio和上层的沟通媒介，它不是sdio slave的controller，sdio IP有自己的controller，有自己的FIFO。

4.2 TI Omap平台下的sdio中断处理实现

截至3.12内核版本，Sdio在TI的Omap Soc下仍然没有实现以中断方式处理sdio事件。具体点说，即mmc子系统软件框架已经考虑了sdio中断，但是针对omap平台的host驱动(omap_hsmmc.c)没有支持，它直接忽略了中断寄存器SD_STAT中的CIRQ。

4.2.1为什么host不支持sdio中断？

究其原因，应该是因为某些平台的host没有swakeup line(smart-idle-wakeup)，比如arm335x。看一下图4.1所示的mmc子系统的大致硬件架构和host的硬件组成就大致明白了。

Host包括hsmmc和PRCM等子模块，其中PRCM提供clock，电源管理时host可能会进入suspend模式，该模式下PRCM不提供clock，所以此时host将无法处理CIRQ中断，除非上层强制host wake up，否则host不会智能恢复正常工作。而如果有swakeup line的时候，当有CIRQ时，通过swakeup line，PRCM将自动供clock。这一点在图1.1中可以看的更清楚。

4.2.2 mmc子系统如何实现sdio中断方式

假设不考虑上述没有swakeup line，进入suspend状态的hsmmc无法在有卡中断时自动开始恢复工作的情况，或者说我们已经找到了某种解决办法后，考虑下host，sdio等mmc子系统模块如何配合实现使用中断方式处理sdio事件。

图4.2表明了在当前mmc子系统软件架构下可行的中断处理方式。

• Sdio 中断

具体的SDIO IP驱动

注册一个irq_handler

Note:

1. 对于sdio中断处理，没有使用tasklet之类的下半部机制，它是直接使用一个高优先级的线程来处理下半部，上半部在host模块中处理，两者的基本思想是一致的，即：ISR中只做最必要的事，如清中断等，耗时的操作放在DSR中去做。不同之处在于Linux标准下半部机制如softirq的执行多数在中断上下文中，而使用线程的方式则一定是在线程上下文中的。
2. Sdio中断处理为什么不直接使用已有的tasklet机制？我觉得1）这还是软件架构的事情。用户注册的irq_handler是调用sdio模块中的接口通过host访问slave，该过程中会sleep导致阻塞，而tasklet是不允许阻塞的；2）单独使用一个线程也可以兼容sdio中断及轮询两者方式处理sdio function事件。

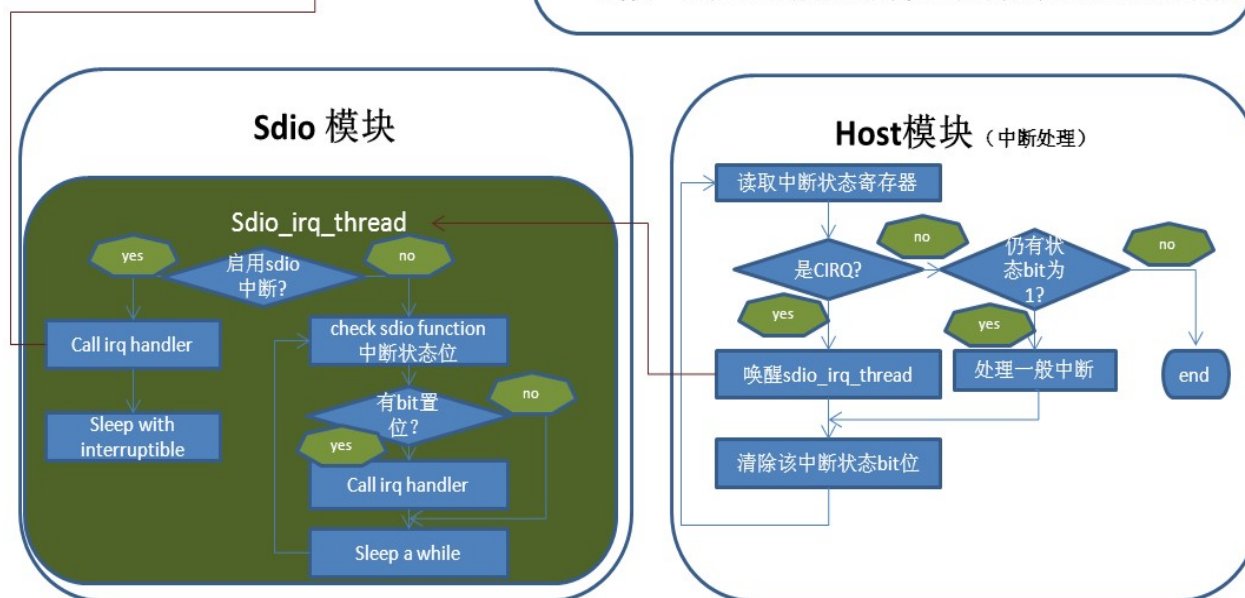


图4.2

请注意图中给出的注意事项。

5. Appendix

5.1 平台初始化流程

5.1.1 Linux内核平台初始化


Linux内核将平台相关的具体数据和模块的特定初始化函数分别放在两个平台相关的文件中。内核提供一个框架，并调用各平台提供的注册函数。

以omap arm33xx为例，板级初始化相关函数放在文件board-am335xevm.c中，由kernel框架调用；板级各模块特定信息放在omap_hwmod_33xx_data.c中，并由omap_hwmod.c中提供的函数进行调用。


注意：现在Linux下已经不用这种方式存储板级信息，取而代之的是device tree，它用一种更简洁的方式来描述这些特定平台信息，从而让kernel代码更加清晰，具体的可以参考<http://blog.csdn.net/21cnbao/article/details/8457546>。

5.1.2 平台环境文件

平台环境文件board-am335xevm.c定义了各模块的初始化函数，以及提供内核初始化必要的平台初始化函数：



```
1 MACHINE_START(AM335XEVM, "xxxx")
2
3 /* Maintainer: Texas Instruments */
4
5 .atag_offset      = 0x100,
6
7 .map_io           = am335x_evm_map_io,
8
9 .init_early       = am33xx_init_early,
10
11 .init_irq         = ti81xx_init_irq,
12
13 .handle_irq       = omap3_intc_handle_irq,
14
15 .timer            = &omap3_am33xx_timer,
16
17 .init_machine     = am335x_evm_init,
18
19 MACHINE_END
```



上面这段代码注册了内核在start_kernel()中会调用的函数，如map_io提供将各IP寄存器地址到内核虚拟地址的转换，init_early()函数由start_kernel()--->setup_arch()--->mdesc->init_early()调用，其余的init_irq,handle_irq()等会在start_kernel()中分别调用。其中init_machine()函数用于初始化各IP模块的device抽象对象，熟悉Linux驱动架构的应该就清楚了，这就是bus,driver,device三剑客之一的device初始化的地方。

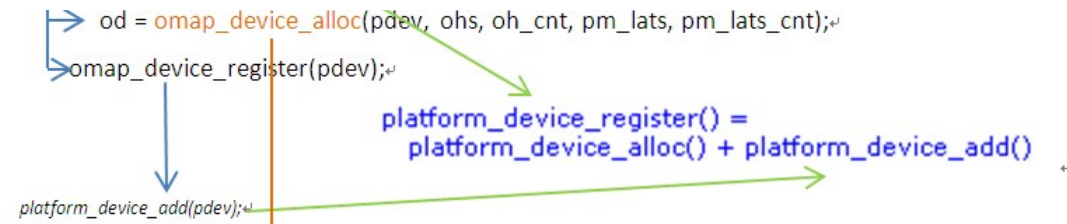
举例来说，init_machine()这个函数被调用流程是：start_kernel()--->rest_init()--->kernel_init()->_do_basic_setup()->_do_initcalls()--->customize_machine()--->init_machine()。

```

am335x_evm_init()
└─> am335x_setup_embedsky_board()
    └─> _configure_device(EMBEDSKY_EVM, embedsky_evm_dev_cfg, PROFILE_NONE);
        └─> mmc0_init()
            └─> setup_pin_mux(mmc0_pin_mux);
                └─> omap2_hsmmc_init(am335x_mmc); platform 资源: board-am335xevm.c
                    └─> static struct omap2_hsmmc_info am335x_mmc[] __initdata = {
                        {
                            .mmc      = 1,
                            .caps      = MMC_CAP_4_BIT_DATA,
                            .gpio_cd    = GPIO_TO_PIN(0, 6),
                            .gpio_wp    = GPIO_TO_PIN(3, 18),
                            .ocr_mask   = MMC_VDD_32_33 | MMC_VDD_33_34, /* 3V3 */
                        },
                        {
                            .mmc      = 0, /* will be set at runtime */
                        },
                        {
                            .mmc      = 0, /* will be set at runtime */
                        },
                        {
                            /* Terminator */
                        },
                    };

omap_init_hsmmc(controllers, controllers->mmc);
omap_device_build()
└─> omap_device_build_ss()
    └─> pdev = platform_device_alloc(pdev_name, pdev_id);
        └─> od = omap_device_alloc(pdev, ohs, oh_cnt, pm_lats, pm_lats_cnt);

```

其中, platform_device 的 resources 资源获取流程(即把 omap_device 中的特定 resources 赋值给通用 platform_device 数据结构中的 resources) 是: ↵

`omap_device_alloc(pdev, ohs, oh_cnt, pm_lats, pm_lats_cnt);` (*omap_device.c*) ↵

`omap_device_fill_resources(od, res);` (*omap_hwmod.c*) ↵

`ret = platform_device_add_resources(pdev, res, res_count);` ↵

而 omap_device 的资源是在 omap_hwmod_33xx_data.c 中定义的: ↵


```

/* mmc0 */
static struct omap_hwmod_irq_info am33xx_mmc0_irqs[] = {
    { .irq = 64 },
    { .irq = -1 }
};

static struct omap_hwmod_dma_info am33xx_mmc0_edma_reqs[] = {
    { .name = "tx", .dma_req = 24, },
    { .name = "rx", .dma_req = 25, },
    { .dma_req = -1 }
};

static struct omap_hwmod_addr_space am33xx_mmc0_addr_space[] = {
    {
        .pa_start = 0x48060100,
        .pa_end   = 0x48060100 + SZ_4K - 1,
        .flags    = ADDR_TYPE_RT,
    },
    {}
};

static struct omap_hwmod_ocp_if am33xx_l4ls__mmc0 = {
    .master = &am33xx_l4ls_hwmod,
    .slave  = &am33xx_mmc0_hwmod,
    .clk    = "mmc0_ick",
    .addr   = am33xx_mmc0_addr_space,
    .user   = OCP_USER_MPU,
};

static struct omap_hwmod_ocp_if *am33xx_mmc0_slaves[] = {
    &am33xx_l4ls__mmc0,
};

static struct omap_mmc_dev_attr am33xx_mmc0_dev_attr = {
    .flags = OMAP_HSMMC_SUPPORTS_DUAL_VOLT,
};

static struct omap_hwmod am33xx_mmc0_hwmod = {
    .name = "mmc1",
    .class = &am33xx_mmc_hwmod_class,
    .clkdm_name = "l4ls_clkdm",
    .mpu_irqs = am33xx_mmc0_irqs,
    .main_clk = "mmc0_fck",
    .sdma_reqs = am33xx_mmc0_edma_reqs,
    .prcm = {
        .omap4 = {
            .clkctrl_offs = AM33XX_CM_PER_MMC0_CLKCTRL_OFFSET,
            .modulemode = MODULEMODE_SWCTRL,
        },
    },
    .dev_attr = &am33xx_mmc0_dev_attr,
    .slaves = am33xx_mmc0_slaves,
    .slaves_cnt = ARRAY_SIZE(am33xx_mmc0_slaves),
};

```

5.1.3 平台数据文件及其调用者

可以看到init_machine()函数在do_initcalls()中处于第三优先级，第一优先级的是core_init()，它会使用omap_hwmod.c提供的相关函数在device初始化前作一些操作。

omap_hwmod_33xx_data.c大部分内容是定义板子各模块的资源信息如clock、irq号、模块寄存器相应的物理地址等。

do_initcalls()时，core_initcall(omap_hwmod_setup_all)---> _setup()(对每个模块依次作此操作)。

所以平台初始化操作除MACHINE_START提供的几个函数外，还有一个地方就是这里的omap_hwmod_setup_all。

以各模块的sysconfig寄存器初始配置为例：

```
core_initcall(omap_hwmod_setup_all);
```

```
    omap_hwmod_for_each(_setup, NULL); //枚举平台omap_hwmod_33xx_data.c中定义的omap_hwmod（每个模块的寄存器设定值）
```

```
_setup
```

+++++++ 以下是clock相关 ++++++

```

_enable
    _enable_sysc
        _set_clockactivity
        _write_sysconfig
            omap_hwmod_write(v, oh, oh->class->sysc->sysc_offs);

```

注意：omap相关模块寄存器默认值一般情况下并非都为0，以mmc的sysconfig为例，其默认值是0x2015。

5.2 slave检测流程

如2.2.5所言，host的device和driver创建并匹配后调用host对应的probe函数：omap_hsmmc_probe()。其中一个重要的步骤是启用一个工作队列进行slave检测并创建后续相关device/driver对象。

Host驱动为了兼容多种host controller，有些操作对于SDIO是不需要的（下面用删除线mark了），但是给SDIO这样的操作，SDIO不应报错。具体流程如下所示：

```

mmc_alloc_host
INIT_DELAYED_WORK(&host->detect, mmc_rescan);
mmc_add_host(mmc);
mmc_start_host(host);

mmc_power_off(host);
    mmc_detect_change(host, 0);
        mmc_schedule_delayed_work(&host->detect, delay);

mmc_rescan
mmc_rescan_try_freq
sdio_reset(host); //使用CMD52设置slave寄存器6 (CCCR) 的RES bit
    mmc_go_idle(host);
    mmc_send_if_cond(host, host->ocr_avail);
mmc_attach_sdio(host)
    mmc_send_io_op_cond(host, 0, &ocr); //使用CMD5命令，获取OCR值 (slave支持的电压范围)
    mmc_attach_bus(host, &mmc_sdio_ops); //设置host的bus操作函数是mmc_sdio_ops
    mmc_select_voltage(host, ocr); //设置电压为选取电压 (slave本身支持的电压范围 & 平台环境选取的电压ocr_avail==ocr_mask) 的最小值
mmc_sdio_init_card(host, host->ocr, NULL, 0);
    mmc_alloc_card(host, NULL); //device_init, 创建card device, 指定为mmc_bus_type
    host->ops->init_card == omap_hsmmc_init_card //没设置，为空
    mmc_send_relative_addr(host, &card->rca); //CMD3, 获取slave的rca

```

```
        mmc_select_card(card);//CMD7,选中rca对应的slave
        sdio_read_cccr(card);//read CCCR
        sdio_read_common_cis(card);// read CIS
        // set high speed, set bus width and so on...

sdio_init_func
        sdio_alloc_func//创建device, 指定为sdio_bus_type
        sdio_read_fbr
sdio_read_func_cis
mmc_add_card//把card device挂载到mmc_bus_type总线
        sdio_add_func//把sdio device挂载到sdio_bus_type总线
```

6. Reference

1. Linux kernel source code(version3.2.0)
2. Linux kernel 最新patch
3. AM335x ARM® Cortex™-A8 Microprocessors(MPUs) Technical Reference Manual
4. SDIO Simplified Specification
5. SD Host Controller Simplified Specification

分类: [general IO driver](#)

标签: [Linux sdio mmc](#)

好文要顶

关注我

收藏该文



[randyqiu](#)

关注 - 0

粉丝 - 1

[+加关注](#)

0

0

« [上一篇: 系统调度和中断处理的异同 \(以Linux MIPS为例\)](#)

» [下一篇: Linux内核中SPI/I2c子系统剖析](#)

posted @ 2014-03-18 10:46 randyqiu Views(4242) Comments(0) Edit 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问网站首页](#)。

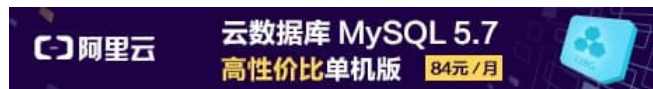
【推荐】超50万VC++源码: 大型组态工控、电力仿真CAD与GIS源码库!

【推荐】华为云7大明星产品0元免费使用

【大赛】2018首届“顶天立地”AI开发者大赛

**最新IT新闻:**

- 美团打车推出垫付功能 未付款订单平台7日内自动垫付
 - 争议EOS: 超级节点，平民玩不起的游戏
 - 有多少iOS应用能年入百万美元? 去年共有2857款
 - “你好好享受童年，我负责改变世界”大佬们正在努力
 - 爱立信获欧盟2.5亿欧元贷款研发5G
- » 更多新闻...

**最新知识库文章:**

- 你可以把编程当做一项托付终身的职业
 - 评审的艺术——谈谈现实中的代码评审
 - 如何高效学习
 - 如何成为优秀的程序员?
 - 菜鸟工程师的超神之路 -- 从校园到职场
- » 更多知识库文章...