SDMMC SDIO eMMC 开发指南

文件标识：RK-KF-YF-121

发布版本：V1.4.0

日期：2024-11-05

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

产品版本

芯片名称	内核版本
全系列	4.4, 4.19, 5.10, 6.1

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	林涛	2017-12-15	初始版本
V1.1.0	林涛	2019-11-12	针对4.19内核修订
V1.1.1	黄莹	2021-05-25	修改格式，增加版权信息
V1.2.0	赵仪峰	2022-09-26	增加SD卡和JTAG复用问题
V1.3.0	林涛	2023-06-05	增加SD卡漏电问题说明
V1.4.0	林涛	2024-11-05	补充模式切换测试和mmc_test兼容性测试项；修正一些属性和说明

目录

1. DTS 配置

1.1 SDMMC 的 DTS 配置说明

1. `max-frequency = <150000000>;`

此配置设置 SD 卡的运行频率，虽然设置为 150MHz，但是还要根据 SD 卡的不同模式进行调整。这部分不需要用户关心，实际运行频率和模块的关系软件会关联。最大不超过 150MHz。

2. `supports-sd;`

此配置标识此插槽为 SD 卡功能，为必须添加项。否则无法初始化 SD 卡。注意4.19内核之后不再使用此配置，改为`no-sdio`与`no-mmc`的组合来表明此卡槽不支持SDIO和MMC，专用为SD功能；

3. `bus-width = <4>;`

此配置标识需要使用 SD 卡的线宽。SD 卡最大支持 4 线模式，如果不配置就模式使用 1 线模式。另外，这个位只支持的数值为 1，4，配置其他数值会认为是非法数值，强制按照 1 线模式进行使用。

4. `cap-mmc-highspeed; cap-sd-highspeed;`

此配置为标识此卡槽支持 highspeed 的 SD 卡。 如果不配置，表示不支持 highspeed 的 SD 卡。

5. 配置使用 SD3.0

首先确保芯片支持 SD3.0 模式，并且需要配置 vqmmc 这一路的 SDMMC 控制器的 IO 电源，并添加如下一些 SD3.0 的速度模式

```
sd-uhs-sdr12: 时钟频率不超过24M
sd-uhs-sdr25: 时钟频率不超过50M
sd-uhs-sdr50: 时钟频率不超过100M
sd-uhs-ddr50: 时钟频率不超过50M，并且采用双沿采样
sd-uhs-sdr104: 时钟频率不超过208M
```

6. 配置 SD 卡设备的 3V3 电源

如果硬件上使用的电源控制引脚是芯片上 SDMMC 控制器默认的电源控制脚：sdmmc_pwren，那么只需要在 pinctrl 上配置为 sdmmc_pwren 的功能脚，并在 sdmmc 节点内引入到 default 的 pinctrl 内即可，例如以 RK312X 为例：

```
sdmmc_pwren: sdmmc-pwren {
    rockchip,pins = <1 RK_PB6 1 &pcfg_pull_default>;
};

pinctrl-0 = <&sdmmc_pwr &sdmmc_clk &sdmmc_cmd &sdmmc_bus4>;
```

如果硬件是使用其他 GPIO 作为 SD 卡设备的 3V3 电源控制引脚，则需要将其定义成 regulator 来使用，并在 sdmmc 的节点内将其引用到 vmmc-supply 内，例如：

```
sdmmc_pwr: sdmmc-pwr {
    rockchip,pins = <7 11 RK_FUNC_GPIO &pcfg_pull_none>;
```

```
};

vcc_sd: sdmmc-regulator {
    compatible = "regulator-fixed";
    gpio = <&gpio7 11 GPIO_ACTIVE_LOW>;
    pinctrl-names = "default";
    pinctrl-0 = <&sdmmc_pwr>;
    regulator-name = "vcc_sd";
    regulator-min-microvolt = <3300000>;
    regulator-max-microvolt = <3300000>;
    startup-delay-us = <100000>;
    vin-supply = <&vcc_io>;
};

&sdmmc {
    vmmc-supply = <&vcc_sd>;
};
```

7. 配置 SD 卡热拔插检测脚

如果检测脚是直接连接到芯片的 SDMMC 控制器的 sdmmc_cd 脚，则请直接将该脚位配置为功能脚，并在 sdmmc 节点的 default 的 pinctrl 内进行引用即可。

如果检测脚是使用其他 GPIO，则需要在 sdmmc 节点内使用 cd-gpios 来进配置，例如

`cd-gpios = <&gpio4 24 GPIO_ACTIVE_LOW>;`

如果使用 GPIO 的检测脚，但是又要求反向检测方式(即 SD 卡插入时检测脚为高电平)，则需要追加

`cd-inverted;`

1.2 SDIO 的 DTS 配置说明

1. `max-frequency = <150000000>;`

此项同 SD 卡的配置，最大运行频率不超过 150Mhz; 如果接入的是SDIO2.0设备，则自动向下兼容为最大 50MHz，如果接入的是SDIO3.0设备则最大支持 150MHz；

2. `supports-SDIO;`

此配置标识此插槽为 SDIO 功能，为必须添加项。否则无法初始化 SDIO 外设。注意4.19内核之后不再使用此配置，改为`no-sd`与`no-mmc`的组合来表明此卡槽不支持SDMMC和MMC，专用为SDIO功能；

3. `bus-width = <4>;`

此配置同 SD 卡功能。

4. `cap-sd-highspeed;`

此配置同 SD 卡功能，作为 SDIO 外设，也有区分是否为 highspeed 的 SDIO 外设。

5. `cap-sdio-irq;`

此配置标识该 SDIO 外设(通常是 Wifi)是否支持 sdio 中断，如果你的外设是 OOB 中断，

请不要加入此项。支持哪种类型的中断请联系 Wifi 原厂确定。

6. `keep-power-in-suspend;`

此配置表示是否支持睡眠不断电，请默认加入该选项。Wifi 一般都有深度唤醒的要求。

7. `mmc-pwrseq = <&sdio_pwrseq>;`

此项是 SDIO 外设(一般是 Wifi)的电源控制。为必须项，否则 Wifi 无法上电工作。请参考下面的例子，晶振时钟和复位-使能的 GPIO 的选择按照实际板级硬件要求进行配置。

```
        sdio_pwrseq:sdio-pwrseq {
                compatible ="mmc-pwrseq-simple";
                clocks = <&rk808 1>;
                clock-names ="ext_clock";
                pinctrl-names ="default";
                pinctrl-0 =<&wifi_enable_h>;
                /*
                * On the module itself this isone of these (depending
                * on the actual cardpopulated):
                * - SDIO_RESET_L_WL_REG_ON
                * - PDN (power down when low)
                */
                reset-gpios = <&gpio0 10GPIO_ACTIVE_LOW>; /* GPIO0_B2 */
        };
```

8. `non-removable;`

此项表示该插槽为不可移动设备且此项为 SDIO 设备必须添加项。

9. `num-slots = <4>;`

此项同 SD 卡的配置。

10. `sd-uhs-sdr104;`

此项配置决定该 SDIO 设备是否支持 SDIO3.0 模式。前提是需要 Wifi 的 IO 电压为 1.8v。

1.3 eMMC 的 DTS 配置

1. `max-frequency = <150000000>;`

此项同 SD 卡的配置，最大运行频率不超过 150Mhz; 若运行在eMMC Highspeed模式则自动向下兼容到50MHz，若运行在eMMC HS200模式最大支持到150MHz。

2. `supports-emmc;`

此配置标识此插槽为 emmc 功能，为必须添加项。否则无法初始化 emmc 外设。注意4.19内核之后不再使用此配置，改为`no-sd`与`no-sdio`的组合来表明此卡槽不支持SDMMC和SDIO，专用为eMMC功能；

3. `bus-width = <4>;`

此配置同 SD 卡功能。

4. `mmc-ddr-1_8v;`

此配置表示支持 50MDDR 模式；

5. `mmc-hs200-1_8v;`

此配置表示支持 HS200 模式；

6. `mmc-hs400-1_8v; mmc-hs400-enhanced-strobe`

此两项配置表示支持 HS400 模式以及 HS400ES 模式，具体芯片支持情况请查阅芯片规格书。

7. `non-removable;`

此项表示该插槽为不可移动设备。 此项为必须添加项。

2. 频率和模式切换

1. 模式切换可以使用上述DTS配置中相关项来实现。
2. 如果希望在不修改固件的情况下，则可以使用动态切换方式。

```
内核6.1版本默认携带了如下提交，其他平台需要打上。
commit 0c11160a3c03bcf41fd1217d5e44e68ff078c54b
Author: Shawn Lin <shawn.lin@rock-chips.com>
Date:   Tue Nov 5 18:05:50 2024 +0800

    mmc: debugfs: Allow more host caps to be moodified
    Change-Id: I2d5df7bc95eaa4a380b5107364aff8df3a759788

commit 478cdcd25cba8be5cfcf56b503b0e070f5aef245
Author: Vincent Whitchurch <vincent.whitchurch@axis.com>
Date:   Fri Sep 29 09:45:09 2023 +0200

    UPSTREAM: mmc: debugfs: Allow host caps to be modified
    Change-Id: I6ea2fc093234785e05d05128556c60ef42da02a4

commit a5a3ea5e45d09cff84c04b515b5b71d4dd76524e
Author: Vincent Whitchurch <vincent.whitchurch@axis.com>
Date:   Fri Sep 29 09:45:08 2023 +0200

    UPSTREAM: mmc: core: Always reselect card type
    Change-Id: Ie80514ade3b267901818f477d74d91cf7c13e103
```

2.1 SD/SDIO动态切换

1. **DTS按照所支持的最大速率模式配置** 。从log识别所需修改的MMC设备ID和地址，以某张tf卡为例

```
[    5.739929] mmc_host mmc1: Bus speed (slot 0) = 400000Hz (slot req
400000Hz, actual 400000HZ div = 0)
[    5.941037] mmc_host mmc1: Bus speed (slot 0) = 198000000Hz (slot req
200000000Hz, actual 198000000HZ div = 0)
[    5.941682] dwmmc_rockchip 2a310000.mmc: Successfully tuned phase to 90
[    5.941705] mmc1: new ultra high speed SDR104 SDHC card at address aaaa
[    5.943229] mmcblk1: mmc1:aaaa SC32G 29.7 GiB
```

我们可以得到三个信息

- MMC设备ID为1, 即**mmc1**
- tf卡的地址为**aaaa**，路径组合即为**mmc1:aaaa**
- 目前运行模式是**sdr104**模式，频率**198MHz**

2. 因为此设备是mmc1，所以执行`cd  /sys/kernel/debug/mmc1`进入对应目录。如果是mmc0则进入mmc0目录，以此类推。
3. 如果希望在当前模式下，仅仅切换频率为100MHz，则使用`echo 100000000 > clock`命令，以此类推。

```
root@rk3576-buildroot:/sys/kernel/debug/mmc1# echo 100000000 > clock
[   86.451685] mmc_host mmc1: Bus speed (slot 0) = 100000000Hz (slot req
100000000Hz, actual 100000000HZ div = 0)
```

4. 切换成High speed模式，默认50MHz

```
#在sys/kernel/debug/mmc1目录依次执行下面命令:
echo $(($(cat caps) | (1 << 7))) > caps
echo on > /sys/bus/mmc/devices/mmc1\:aaaa/power/control
echo auto > /sys/bus/mmc/devices/mmc1\:aaaa/power/control
echo $(($(cat caps) & ~(1 << 19))) > caps
echo on > /sys/bus/mmc/devices/mmc1\:aaaa/power/control

#可以看到切换了
[   78.304578] mmc_host mmc1: Bus speed (slot 0) = 400000Hz (slot req
400000Hz, actual 400000HZ div = 0)
[   78.501126] mmc_host mmc1: Bus speed (slot 0) = 50000000Hz (slot req
50000000Hz, actual 50000000HZ div = 0)

#执行下面命令，确认切换成功了
grep timing ios
timing spec:       2 (sd high-speed) #High speed模式了
```

5. 切换成Default speed模式，默认25MHz

```
#在sys/kernel/debug/mmc1目录依次执行下面命令:
echo $(($(cat caps) | (1 << 7))) > caps
echo on > /sys/bus/mmc/devices/mmc1\:aaaa/power/control
echo auto > /sys/bus/mmc/devices/mmc1\:aaaa/power/control
echo $(($(cat caps) & ~(1 << 2))) > caps
echo on > /sys/bus/mmc/devices/mmc1\:aaaa/power/control

#可以看到切换了
[  287.590885] mmc_host mmc1: Bus speed (slot 0) = 400000Hz (slot req
400000Hz, actual 400000HZ div = 0)
[  287.784531] mmc_host mmc1: Bus speed (slot 0) = 25000000Hz (slot req
25000000Hz, actual 25000000HZ div = 0)

#执行下面命令，确认切换成功了
grep timing ios
timing spec:    0 (legacy) #Defalt speed模式了
```

2.2 eMMC动态切换

1. **DTS按照所支持的最大速率模式配置**。从log识别所需修改的MMC设备ID和地址，以RK3588 eMMC为例

```
[    2.260445] mmc0: Command Queue Engine enabled
[    2.260458] mmc0: new HS400 Enhanced strobe MMC card at address 0001
[    2.260740] mmcblk0: mmc0:0001 BJTD4R 29.1 GiB
```

我们可以得到三个信息

- MMC设备ID为0, 即**mmc0**
- eMMC设备的地址为**0001**，路径组合即为**mmc0:0001**
- 目前运行模式是**HS400**模式，平台eMMC默认频率**198MHz**

2. 因为此设备是mmc0，所以执行`cd  /sys/kernel/debug/mmc0`进入对应目录。如果是mmc1则进入mmc1目录，以此类推。
3. 如果希望在当前模式下，仅仅切换频率，则使用`echo 100000000 > clock`命令，以此类推。

```
#先确认默认频率和模式
root@rk3588-buildroot:/sys/kernel/debug/mmc0# grep clock ios
clock:          200000000 Hz
actual clock:   198000000 Hz  #默认198MHz
root@rk3588-buildroot:/sys/kernel/debug/mmc0# grep timing ios
timing spec:    10 (mmc HS400 enhanced strobe) #HS400es模式

#开始切换频率
root@rk3588-buildroot:/sys/kernel/debug/mmc0# echo 100000000 > clock
root@rk3588-buildroot:/sys/kernel/debug/mmc0# grep clock ios
clock:          100000000 Hz
actual clock:   100000000 Hz  #切换成100MHz了
root@rk3588-buildroot:/sys/kernel/debug/mmc0# grep timing ios
timing spec:    10 (mmc HS400 enhanced strobe) #仍然HS400es模式
```

4. 切换成HS200模式，默认198MHz

```
#在sys/kernel/debug/mmc0目录依次执行下面命令:
echo $(($(cat caps) | (1 << 7))) > caps
echo on > /sys/bus/mmc/devices/mmc0\:0001/power/control
echo auto > /sys/bus/mmc/devices/mmc0\:0001/power/control
echo $(($(cat caps2) & ~(1 << 15))) > caps2
echo on > /sys/bus/mmc/devices/mmc0\:0001/power/control

#执行下面命令，确认切换成功了
grep timing ios
timing spec:       9 (mmc HS200) #HS200模式了
```

5. 切换成High speed模式，默认50MHz

```
#在sys/kernel/debug/mmc0目录依次执行下面命令：
echo $(($(cat caps) | (1 << 7))) > caps
echo on > /sys/bus/mmc/devices/mmc0\:0001/power/control
echo auto > /sys/bus/mmc/devices/mmc0\:0001/power/control
echo $(($(cat caps) & ~(1 << 1))) > caps
echo on > /sys/bus/mmc/devices/mmc0\:0001/power/control

#执行下面命令，确认切换成功了
grep timing ios
timing spec:    1 (mmc high-speed) #High speed模式了
```

2.3 其他说明

除了频率和模式切换，目前还支持线宽切换(多线宽向下切换可能带来模式的自动降级)、CMD23支持开关、CQE和DCMD开关，以mmc0:0001设备为例：

```
#在sys/kernel/debug/mmc0目录依次执行下面命令：
echo $(($(cat caps) | (1 << 7))) > caps
echo on > /sys/bus/mmc/devices/mmc0\:0001/power/control
echo auto > /sys/bus/mmc/devices/mmc0\:0001/power/control

#下面的切换配置按需选择
echo $(($(cat caps) & ~(1 << 6))) > caps  #去除8bit支持
echo $(($(cat caps) | (1 << 6))) > caps  #添加8bit支持
echo $(($(cat caps) & ~(1 << 0))) > caps  #去除4bit支持
echo $(($(cat caps) | (1 << 0))) > caps  #添加4bit支持
echo $(($(cat caps) & ~(1 << 30))) > caps  #去除cmd23支持
echo $(($(cat caps) | (1 << 30))) > caps  #添加cmd23支持
echo $(($(cat caps) & ~(1 << 30))) > caps  #去除cmd23支持
echo $(($(cat caps) | (1 << 30))) > caps  #添加cmd23支持
echo $(($(cat caps2) & ~(1 << 23))) > caps2  #去除cqe支持
echo $(($(cat caps2) | (1 << 23))) > caps2  #添加cqe支持
echo $(($(cat caps2) & ~(1 << 24))) > caps2  #去除cqe dcmd支持
echo $(($(cat caps2) | (1 << 24))) > caps2  #添加cqe dcmd支持

#再执行这句
echo on > /sys/bus/mmc/devices/mmc0\:0001/power/control

#确认ios信息，查看输出结果
cat ios
```

以上这些caps和caps2支持的bit信息可查询内核include/linux/mmc/host.h文件。目前支持的caps和caps2为：

caps	caps2
MMC_CAP_AGGRESSIVE_PM	MMC_CAP2_HS400_1_8V
MMC_CAP_SD_HIGHSPEED	MMC_CAP2_HS400_1_2V
MMC_CAP_MMC_HIGHSPEED	MMC_CAP2_HS400_ES
MMC_CAP_UHS	MMC_CAP2_HS200_1_8V_SDR
MMC_CAP_DDR	MMC_CAP2_HS200_1_2V_SDR
MMC_CAP_4_BIT_DATA	MMC_CAP2_CQE
MMC_CAP_8_BIT_DATA	MMC_CAP2_CQE_DCMD
MMC_CAP_CMD23	

如果项目这些还有额外需要支持的配置，可以模仿前述`commit 0c11160a3c03bcf41fd1217d5e44e68ff078c54b`提交自行追加。

3. mmc_test兼容性测试

推荐依赖SD卡的产品导入SD卡专项兼容性测试中。

3.1 内核配置确认

```
请确认添加了如下配置CONFIG_MMC_TEST=y:
Device Drivers  --->
   <*> MMC/SD/SDIO card support  --->
   [*]   MMC host test driver

如果选为模块形式，请在开机后insmod mmc_test.ko
```

3.2 使用方法

```
1. 列出所有的mmc设备
ls /sys/bus/mmc/devices/

2. 确定要测试的mmc设备，以某sd卡为例，从log查询其路径组合为
mmc1:aaaa

3. 将sd卡移除出系统
echo 'mmc1:aaaa' > /sys/bus/mmc/drivers/mmcblk/unbind

4. 将sd卡绑定在mmc_test
echo 'mmc1:aaaa' > /sys/bus/mmc/drivers/mmc_test/bind
```

```
5. 此时可以看到如下log
mmc_test mmc1:aaaa: Card claimed for testing.

6. 查看可测试项目
cat /sys/kernel/debug/mmc1/mmc1:aaaa/testlist

7. 根据测试需要选择测试项目，N的含义请查看下方测试说明
echo N > /sys/kernel/debug/mmc1/mmc1:aaaa/test
```

3.3 输出结果

```
mmc1: Result: OK                    ---测试完成
mmc1: Result: FAILED                ---测试失败
mmc1: Result: UNSUPPORTED (by host) ---控制器配置不支持
mmc1: Result: UNSUPPORTED (by card) ---卡不支持
```

3.4 测试说明

N数值	测试名	说明
0	Run all tests	运行所有的测试
1	Basic write (no data verification)	写测试，不校验数据
2	Basic read (no data verification)	读测试，不校验数据
3	Basic write (with data verification)	写测试，校验数据
4	Basic read (with data verification)	读测试，校验数据
5	Multi-block write	多块写
6	Multi-block read	多块读
7	Power of two block writes	偶次方块写入
8	Power of two block reads	偶次发块读出
9	Weird sized block writes	各种奇怪的块数写入
10	Weird sized block reads	各种奇怪的块数读出
11	Badly aligned write	不对齐地址单块写入
12	Badly aligned read	不对齐地址单块读出
13	Badly aligned multi-block write	不对齐地址多块写入
14	Badly aligned multi-block read	不对齐地址多块读出
15	Proper xfer_size at write (start failure)	故意创造一个错误的传输类型，比如用单块传输命令来执行一个多块写入
16	Proper xfer_size at read (start failure)	故意创造一个错误的传输类型，比如用单块传输命令来执行一个多块读出
17	Proper xfer_size at write (midway failure)	与15相同，只是在多块传输中插入写错误
18	Proper xfer_size at read (midway failure)	与16相同，只是在多块传输中插入读错误
19	Highmem write	用High memory page来执行单块写数据
20	Highmem read	用High memory page来执行单块读数据
21	Multi-block highmem write	用High memory page来执行多块写数据
22	Multi-block highmem read	用High memory page来执行多块读数据
23	Best-case read performance	执行512K顺序读，地址连续，不启用sg方式
24	Best-case write performance	执行512K顺序写，地址连续，不启用sg方式

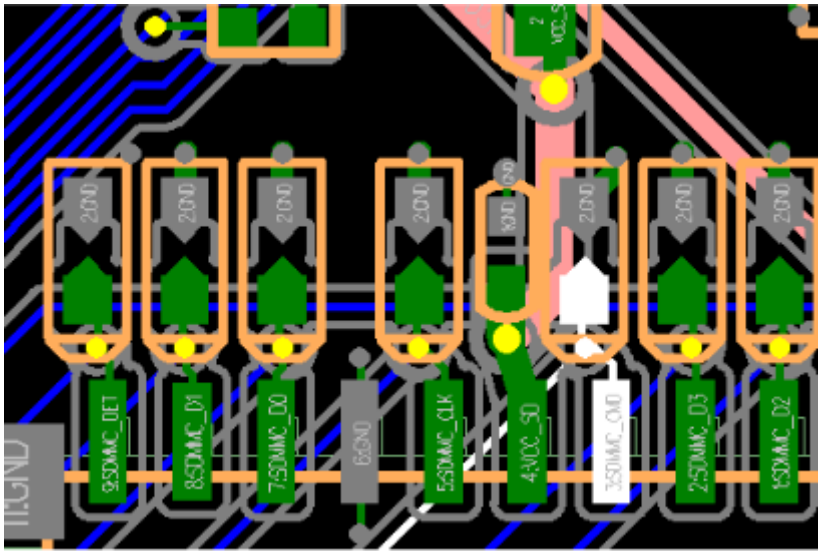
N数值	测试名	说明
25	Best-case read performance into scattered pages	执行512K顺序读，启用sg方式
26	Best-case write performance from scattered pages	执行512K顺序读，启用sg方式
27	Single read performance by transfer size	单块读性能
28	Single write performance by transfer size	单块写性能
29	Single trim performance by transfer size	单块擦除性能
30	Consecutive read performance by transfer size	持续读性能
31	Consecutive write performance by transfer size	持续写性能
32	Consecutive trim performance by transfer size	持续擦除性能
33	Random read performance by transfer size	随机读性能
34	Random write performance by transfer size	随机写性能
35	Large sequential read into scattered pages	用sg方式的大量连续读测试
36	Large sequential write from scattered pages	用sg方式的大量连续写测试
37	Write performance with blocking req 4k to 4MB	测试4K到4MB下不同传输长度的阻塞写性能
38	Write performance with non-blocking req 4k to 4MB	测试4K到4MB下不同传输长度的非阻塞写性能
39	Read performance with blocking req 4k to 4MB	测试4K到4MB下不同传输长度的阻塞读性能
40	Read performance with non-blocking req 4k to 4MB	测试4K到4MB下不同传输长度的非阻塞读性能
41	Write performance blocking req 1 to 512 sg elems	测试1到512个sg下的阻塞写性能
42	Write performance non-blocking req 1 to 512 sg elems	测试1到512个sg下的非阻塞写性能
43	Read performance blocking req 1 to 512 sg elems	测试1到512个sg下的阻塞读性能

N数值	测试名	说明
44	Read performance non-blocking req 1 to 512 sg elems	测试1到512个sg下的非阻塞写性能
45	Reset test	重置测试项
46	Commands during read - no Set Block Count (CMD23)	读过程中发送各种命令(非CMD23方式)
47	Commands during write - no Set Block Count (CMD23)	写过程中发送各种命令(非CMD23方式)
48	Commands during read - use Set Block Count (CMD23)	读过程中发送各种命令(CMD23方式)
49	Commands during write - use Set Block Count (CMD23)	写过程中发送各种命令(CMD23方式)
50	Commands during non-blocking read - use Set Block Count (CMD23)	非阻塞读过程中发送各种命令(CMD23方式)
51	Commands during non-blocking write - use Set Block Count (CMD23)	非阻塞写过程中发送各种命令(CMD23方式)

4. 常见问题排查

4.1 硬件问题分析

1. SD 卡



从左到右依次是：

DET ---- 检测脚

DATA1 ---- 数据线

DATA0

GND

CLK ---- 时钟

VCC_SD ---- SD 卡供电电源

VCCIO_SD ---- 数据线的 IO 供电电源

CMD ---- 命令线

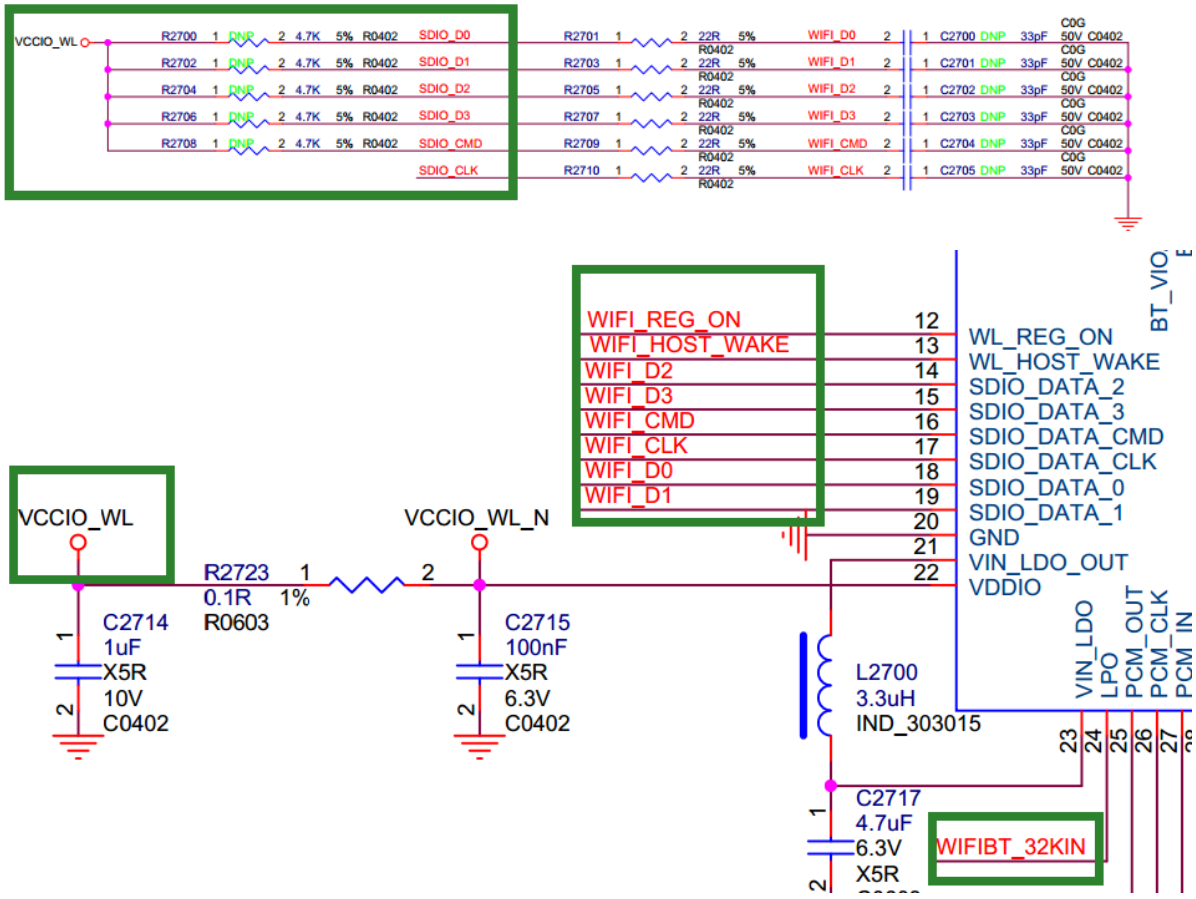
DATA3

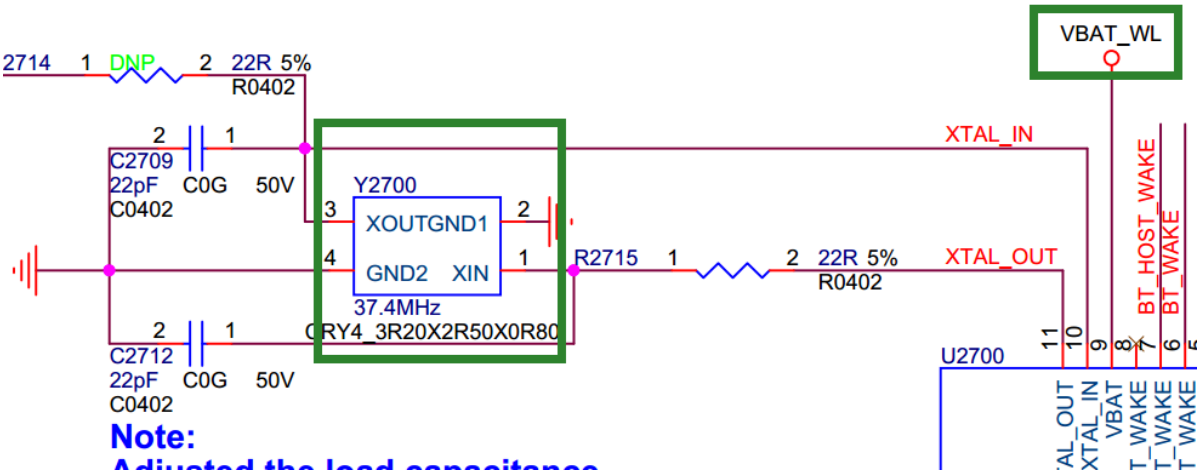
DATA2

除了 DET/CLK/GND 外，其它的 DATA0-3/VCC_SD/VCCIO_SD/CMD 必须都为 3.3v 左右，最小不能低于 3v；DET 脚插入为低，拔出为高； DATA0-3/CMD 的电压都是 VCCIO_SD 供给的，所以 DATA0-3/CMD 必须跟 VCCIO_SD 保持一致，而 VCC_SD 和 VCCIO_SD 要保持一致（NOTE: SD 3.0，要求 VCCIO_SD 为 1.8v）；

如果 VCC_SD/VCCIO_SD 的电源是长供电，那么请保证 VCC_SD 和 VCCIO_SD 在卡拔插时不会有塌陷；

2. SDIO





首先看下硬件：主要的部分都在绿色方框内

WIFI_D0~3：数据线，平时为高，电压取决于 VCCIO_WL 的电压；

WIFI_CMD：命令线，平时为高，电压取决于 VCCIO_WL 的电压；

WIFI_CLK：时钟，平时为低，电压取决于 VCCIO_WL 的电压；

VBAT_WL：WIFI 模组供电电源，一直都为高，供电需打印 3.3v；

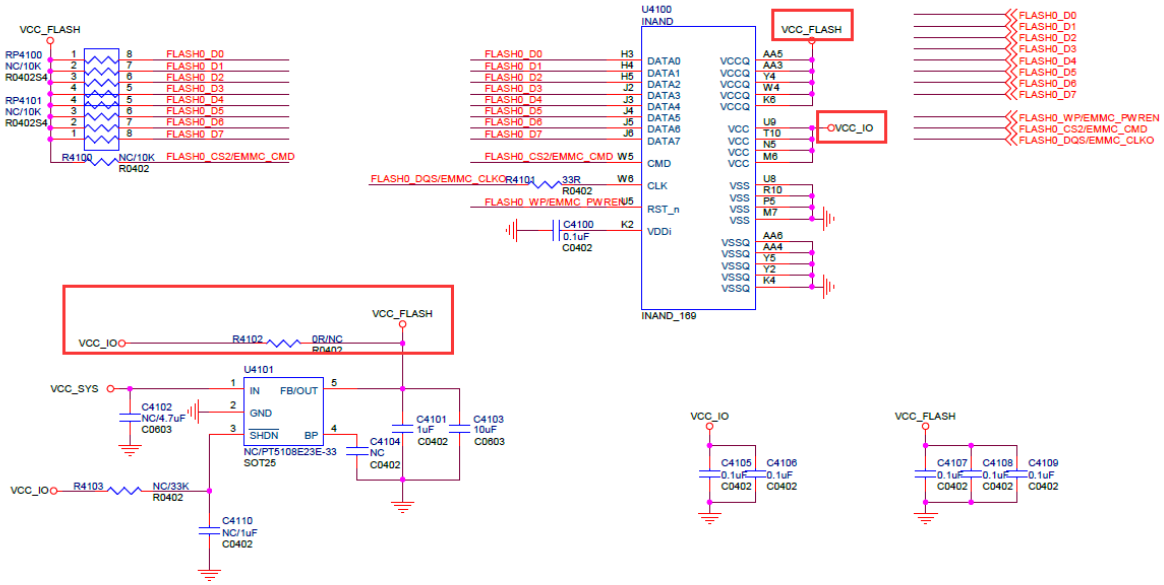
VCCIO_WL：给 DATA/CMD/CLK 的 IO 供电电源，可以为 3.3 或者 1.8v，但 SDIO3.0

必须为 1.8v；

WIFI_REG_ON: 正常工作时为 3.3v，WiFi 关闭时为 0v；

两个晶振：32K 和 26M/37.4M,正常工作时都会有波形输出；

3. eMMC



eMMC 有效电压的组合：

Table 199 — *e*•MMC voltage combinations

		V_{CCQ}		
		1.1 V-1.3 V	**1.70 V–1.95 V**	**2.7 V–3.6 V**
V_{CC}	**2.7 V–3.6 V**	Valid	Valid	Valid (1)
	1.7 V–1.95 V	Valid	Valid	NOT VALID

NOTE 1 V_{CCQ} (I/O) 3.3 V range is not supported in either HS200 or HS400 devices

VCC_FLASH 对应 VCC；

VCC_IO 对应 VCCQ；

确保 eMMC_CMD/DATA0~7/VCC_IO 电压都一致（1.8 或 3.3v）；

确保 VCC_FLAHS/VCC_IO 的电压在开机和运行时或者休眠唤醒时必须保持稳定、不能有塌陷或者纹波过大的情况；

有条件的话，测下 clk 和 cmd 以及 data 的波形质量，确保波形正常；

4.2 波形分析

下图是 SD 卡识别模式时的波形时序图（sdio、emmc 一样）

简单说一下识别 SD 卡的方式：主控发出 48clk 并携带 48bit 的数据发给 SD 卡，而 SD 卡要回应给主控 48clk 加 48bit 的数据；如下图：

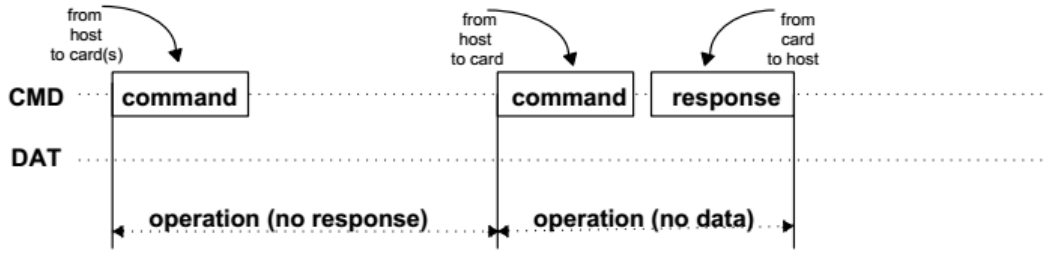
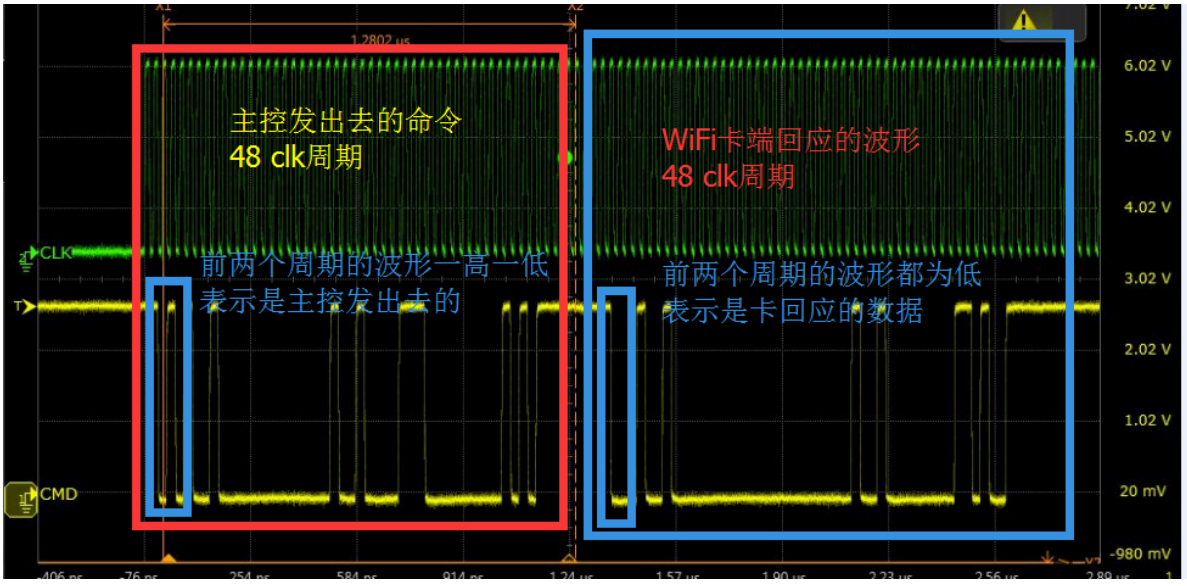


Figure 3-4: "no response" and "no data" Operations



绿色：SDMMC_CLK

黄色：SDMMC_CMD: SDMMC_CMD 空闲时一直处于高电平；

主控发出的波形：当最开始的两个电平有一高一低时，是主控发出去的命令；

SD 卡响应的波形： 当最开始的两个电平有连续的两个低电平是表示卡端有响应；

其次主控和响应一般包含 48 个 bit 的数据，所以 48 个 clk 为一个完整的包。要确认的就是：主控发出去命令包后,SD 卡端是否有响应。

4.3 LOG 分析

1. 正确识别 SD 卡的 LOG

```
[  293.194013] mmc1: new high speed SDXC card at address 59b4
[  293.198185] mmcblk1: mmc1:59b4 00000 59.6 GiB
[  293.204351]  mmcblk1: p1
```

如果在内核看到这样的打印，说明 SD 卡已经被正确识别，并且已经有一个可用的分区 p1。

如果在用户界面看不到 SD 卡设备或者设备不可使用，请排查用户态磁盘守护进程，如 vold。

另外可手动验证分区是否可以使用

`mount -t vfat /dev/block/mmcblk1p1 /mnt`

或者

`mount -t vfat /dev/block/mmcblk1 /mnt`

然后到 mnt 目录下看下是否有 SD 卡里面的文件

2. 开机不读卡,运行时拔插 OK：大概率时电源问题

例如：拔掉所有电源，发现查着 HDMI 发现有漏电到 VCC_SD 卡里面；或者使用外接电源进行测试。

3. 挂载失败：

如果已经看到(1)中的 LOG，但是看到如下挂载失败的 LOG

```
[ 2229.405694] FAT-fs (mmcblk1p1): bogus number of reserved sectors
[ 2229.405751] FAT-fs (mmcblk1p1): Can't find a valid FAT filesystem
```

请格式化 SD 卡为 FAT32 文件系统；

或者 NTFS: make menuconfig 选择 NTFS 文件系统的支持即可；

4. 概率性不识别：

```
mmc1: new high speed SD card at address b368
mmcblk1: mmc1:b368 SMI   486 MiB
[mmc1] Data transmission error !!!!  MINTSTS: [0x00002000]
dwmmc_rockchip ff0c0000.rksdmmc: data FIFO error (status=00002000)
mmcblk1: error -110 sending status command, retrying
need_retune:0,brq->retune_retry_done:0.
```

降频和增加卡检测延时增强电源稳定性，如果降频 OK 的话，请检查硬件 layout；

```
&sdmmc {
    card-detect-delay = <1200>;
}
```

5. TF 卡已经 mount，但不能访问 TF 卡目录，看起来是卡文件系统问题，但卡在 Windows 下可以访问。

请尝试使用 fsck 对 TF 卡做修复。

6. 硬件问题，io 电压异常

```
Workqueue: kmmcd mmc_rescan
[<c0013e24>] (unwind_backtrace+0x0/0xe0) from [<c001172c>] (show_stack+0x10/0x14)
[<c001172c>] (show_stack+0x10/0x14) from [<c04fa444>] (dw_mci_set_ios+0x9c/0x21c)
[<c04fa444>] (dw_mci_set_ios+0x9c/0x21c) from [<c04e7748>]
(mmc_set_chip_select+0x18/0x1c)
[<c04e7748>] (mmc_set_chip_select+0x18/0x1c) from [<c04ebd5c>]
(mmc_go_idle+0x94/0xc4)
[<c04ebd5c>] (mmc_go_idle+0x94/0xc4) from [<c0748d80>]
(mmc_rescan_try_freq+0x54/0xd0)
[<c0748d80>] (mmc_rescan_try_freq+0x54/0xd0) from [<c04e85d0>]
(mmc_rescan+0x2c4/0x390)
[<c04e85d0>] (mmc_rescan+0x2c4/0x390) from [<c004d738>]
(process_one_work+0x29c/0x458)
[<c004d738>] (process_one_work+0x29c/0x458) from [<c004da88>]
(worker_thread+0x194/0x2d4)
[<c004da88>] (worker_thread+0x194/0x2d4) from [<c0052fb4>] (kthread+0xa0/0xac)
[<c0052fb4>] (kthread+0xa0/0xac) from [<c000da98>] (ret_from_fork+0x14/0x3c)
1409..dw_mci_set_ios:  wait for unbusy timeout..... STATUS = 0x306 [mmc1]
```

请检查 CMD 线与 DATA 的电压是否在空载状态下为高电平。并且检测 IO 电压是否过低，以及 IO 电压与电源域的配置是否一致。如果是 SDIO 接口，建议排查 VCCIO_WL 电压，VBAT_WL 和 WIFI_REG_ON 以及晶振是否正常。另可以尝试排查走线太长导致波形质量很差，降频进行测试。

4.4 其他问题

1. u-boot下SD卡1线模式工作正常，4线模式工作报错

大部分SOC的SD卡都会和JTAG复用，在没有插卡时，SOC会自动切换IO到JTAG功能。

EVB参考板会按SOC要求设计SD DET低电平为有插卡，个别客户会修改原理图，设计SD DET高电平为有插卡，这时SOC会误判，把IO切换到JTAG功能。

解决办法： 查找对应芯片GRF寄存器定义，配置force_jtag为Disable，关闭JTAG IO自动切换功能。

RV1126参考代码：

```
diff --git a/arch/arm/mach-rockchip/rv1126/rv1126.c b/arch/arm/mach-
rockchip/rv1126/rv1126.c
index 311310d3f2..29b694df9c 100644
--- a/arch/arm/mach-rockchip/rv1126/rv1126.c
+++ b/arch/arm/mach-rockchip/rv1126/rv1126.c
@@ -544,6 +544,9 @@ void board_debug_uart_init(void)
 #ifndef CONFIG_TPL_BUILD
 int arch_cpu_init(void)
 {
+       struct rv1126_grf * const grf = (void *)GRF_BASE;
+
+       writel(0x00100000, &grf->iofunc_con3);
        /*
         * CONFIG_DM_RAMDISK: for ramboot that without SPL.
```

```
         */
```

2. SD卡漏电导致卡工作异常

当SD模块的IO电源不可控时，若插入卡后，卡的3V3供电还未提供前，此时会从IO上产生漏电倒灌到SD卡，使得SD卡的3V3供电有半电平，易使得个别卡概率性异常，表现为平台兼容性差。因此为了解决漏电问题，驱动会在插入卡时，先将SD卡IO设置为下拉，释放掉漏电后再对SD卡的3V3电源进行上电。当SD卡上电完成之后再恢复SD卡的IO为SD功能脚并设置上拉，从而避免漏电可能导致的一系列异常。对SD卡兼容性要求较高的客户，请确认SDK内核的drivers/mmc/host/dw_mmc.c文件包含了提交“mmc: dw_mmc: Add normal and idle pinctrl control”，并参考下面的例子，修改自己的DTS节点，新增pinctrl的idle模式，需要根据实际板子的使用情况，配置所需的clk、cmd和data线：

```
--- a/arch/arm/boot/dts/rv1126-evb-v10.dtsi
+++ b/arch/arm/boot/dts/rv1126-evb-v10.dtsi
    wireless_wlan: wireless-wlan {
@@ -104,9 +99,14 @@ sdmmc_pwren: sdmmc-pwren {
            rockchip,pins = <0 RK_PA1 RK_FUNC_GPIO &pcfg_pull_none>;
        };

+       sdmmc0_idle_pins: sdmmc-idle-pins {
+           rockchip,pins =
+               <3 RK_PA2 RK_FUNC_GPIO &pcfg_pull_down>,
+               <3 RK_PA3 RK_FUNC_GPIO &pcfg_pull_down>,
+               <3 RK_PA4 RK_FUNC_GPIO &pcfg_pull_down>,
+               <3 RK_PA5 RK_FUNC_GPIO &pcfg_pull_down>,
+               <3 RK_PA6 RK_FUNC_GPIO &pcfg_pull_down>,
+               <3 RK_PA7 RK_FUNC_GPIO &pcfg_pull_down>;
+       };
    };

@@ -140,21 +140,18 @@ &sdio {
 };

 &sdmmc {
    max-frequency = <200000000>;
    no-sdio;
    no-mmc;
    bus-width = <4>;
    cap-mmc-highspeed;
    cap-sd-highspeed;
    disable-wp;
-   pinctrl-names = "default";
+   pinctrl-names = "normal", "idle";
    pinctrl-0 = <&sdmmc0_clk &sdmmc0_cmd &sdmmc0_det &sdmmc0_bus4>;
+   pinctrl-1 = <&sdmmc0_idle_gpios &sdmmc0_det>;
    vmmc-supply = <&vcc3v3_sd>;
    vqmmc-supply = <&vccio_sd>;
    status = "okay";
 };
```