# NAND 模块
# 开发指南

**版本号: 0.3**
**发布日期: 2022.05.11**

# 版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
|---|---|---|---|
| 0.1 | 2021.11.23 | AWA1087 | 1. 建立初始版本 |
| 0.2 | 2021.12.02 | AWA1087 | 1. 新增 dts 配置 boot_crc 说明 2. 新增 FAQ 常见问题，其他章节 3. 新增文档适用平台 H313 |
| 0.3 | 2022.05.11 | AWA332 | 1. 新增文档适用平台 H618<br>2. 增加 pwroff-gpios 使用说明 |

# 目　录

# 1 前言

## 1.1 文档简介

本文档主要介绍 raw nand 全志 nftl 方案的开发使用，包括了对外接口，常用的调试节点和常见问题。

## 1.2 目标读者

nand 开发与维护人员、nand 驱动开发人员

## 1.3 适用范围

表 1-1: 适用产品列表

| 产品名称 | 内核版本 | 驱动文件 |
| --- | --- | --- |
| A100 | Linux-4.9 | modules/nand/common1/* |
| A133 | Linux-4.9 | modules/nand/common1/* |
| A133 | Linux-5.4 | modules/nand/common1/* |
| A133 | Linux-5.15 | bsp/modules/nand/common1/* |
| T507 | Linux-4.9 | modules/nand/common1/* |
| H616 | Linux-4.9 | modules/nand/common1/* |
| H313 | Linux-4.9 | modules/nand/common1/* |
| H618 | Linux-5.4 | modules/nand/common1/* |
| T113 | Linux-5.4 | modules/nand/common1/* |

# 2 模块介绍

## 2.1 模块功能介绍

NAND 存储驱动负责系统代码、文件等各类数据的存储，内部有 NDFC 控制器操作，NFTL 算法（包括垃圾回收、磨损平衡、地址映射等），屏蔽闪存设备读写擦的特性。

## 2.2 相关术语介绍

### 2.2.1 硬件术语

无

### 2.2.2 软件术语

| 术语 | 解释 |
| --- | --- |
| NFTL | NAND Flash Translation Layer |
| gc | Garbage Collection 垃圾回收 |
| wl | Wear Leveling 磨损平衡 |

## 2.3 模块配置介绍

### 2.3.1 sys_config.fex 配置说明

无

### 2.3.2 Device Tree 配置说明

设备树中存在的是该类芯片所有平台的模块配置，设备树文件的路径为：kernel/linux-4.9/arch/armxx/boot/dts/sunxi/CHIP.dtsi(CHIP 为研发代号，如 sun50iw10p1 等)。

```
nand0:nand0@04011000 {
            compatible = "allwinner,sun50iw10-nand";
            device_type = "nand0";
            reg = <0x0 0x04011000 0x0 0x1000>;/* nand0 */
            interrupts = <GIC_SPI 38 IRQ_TYPE_LEVEL_HIGH>;
            clocks = <&clk_pll_periph0x2>,<&clk_nand0>,<&clk_nand1>;
            pinctrl-names = "default", "sleep";
            pinctrl-0 = <&nand0_pins_a &nand0_pins_b>;
            pinctrl-1 = <&nand0_pins_c>;
            nand0_regulator1-supply = <&reg_dcdc1>;
            nand0_regulator2-supply = <&reg_eldo1>;
            nand0_cache_level = <0x55aaaa55>;
            nand0_flush_cache_num = <0x55aaaa55>;
            nand0_capacity_level = <0x55aaaa55>;
            nand0_id_number_ctl = <0x55aaaa55>;
            nand0_print_level = <0x55aaaa55>;
            nand0_p0 = <0x55aaaa55>;
            nand0_p1 = <0x55aaaa55>;
            nand0_p2 = <0x55aaaa55>;
            nand0_p3 = <0x55aaaa55>;
            chip_code = "sun50iw10";
            status = "okay";
        };
```

| 配置项 | 功能 |
| --- | --- |
| pinctrl-0 | 默认的引脚配置状态，详见 kernel/linux-4.9/arch/armxx/boot/dts/sunxi/CHIP-pinctrl.dtsi，用户可通过 pinctrl 中修改引脚驱动能力，上下拉，一般 nand CE、RB 引脚默认上拉 |
| pinctrl-1 | 休眠时的引脚状态，一般不需要用户修改 |
| nand0_regulator1-supply | VCC-NAND 电源配置，根据实际硬件原理图修改，可配置在对应的 board.dts 中 |
| nand0_regulator2-supply | VCCQ 电源配置，根据实际硬件原理图修改，可配置在对应的 board.dts 中 |
| nand0_cache_level | 默认 0x55aaaa55，用于调整算法 cache 数量，不建议用户修改 |
| nand0_capacity_level | 默认 0x55aaaa55，每个分区的保留比例是十分之一；改为 1：1. 每个分区的保留比例由十分之一改为十三分之一。2. 重负载下随机写速度会降低。不建议用户修改 |
| nand0_flush_cache_num | 默认 0x55aaaa55，不建议用户修改 |
| nand0_id_number_ctl | 默认 0x55aaaa55，修改 two plane，interleave、dual channel 配置，不建议用户修改 |
| nand0_print_level | 默认 0x55aaaa55，修改算法打印等级，不建议用户修改 |
| nand0_px | 配合 nand0_id_number_ctl 一起使用，修改 flash 频率，two plane，interleave、dual channel 配置，不建议用户修改 |
| chip_code | 绑定平台，不建议用户修改 |
| boot_crc | “disabled” 关闭启动检测逻辑页 crc 功能，默认打开 |

## 2.3.3 board.dts 配置说明

board.dts 用于保存每一个板级平台的设备信息（如 demo 板，perf1 板等），里面的配置信息会覆盖上面的 Device Tree 默认配置信息。

```
&nand0 {
    nand0_regulator1-supply = <&reg_dcdc1>;
    nand0_regulator2-supply = <&reg_eldo1>;
    pwroff-gpios = <&pio PH 5 (GPIO_ACTIVE_LOW | GPIO_PULL_UP)>;
};
 备注：VCC-NAND/VCCQ电源配置，根据实际硬件原理图修改，用户需要在方案上检查配置是否符合硬件原理图
 pwroff-gpios:用于掉电时候停止操作flash。当pwroff-gpios接受到底电平信号的时候，host会立即停止操作flash
    ,具体使用哪个pin由实际电路决定。
 使用这个功能需要以下几个条件：
                a.驱动支持pwroff-gpios，驱动版本号v2.06 2022-03-15 15:15
                c.kernel menuconfig使能，具体见下面kernel menuconfig 配置说明
                b.硬件电路上有支持监测到掉电的时候输出低电平的功能，具体找硬件负责人
```

## 2.3.4 kernel menuconfig 配置说明

```
Device Drivers->Device Drivers->Block devices->The sunxi nand driver-->
choice spinand or rawnand-->
```

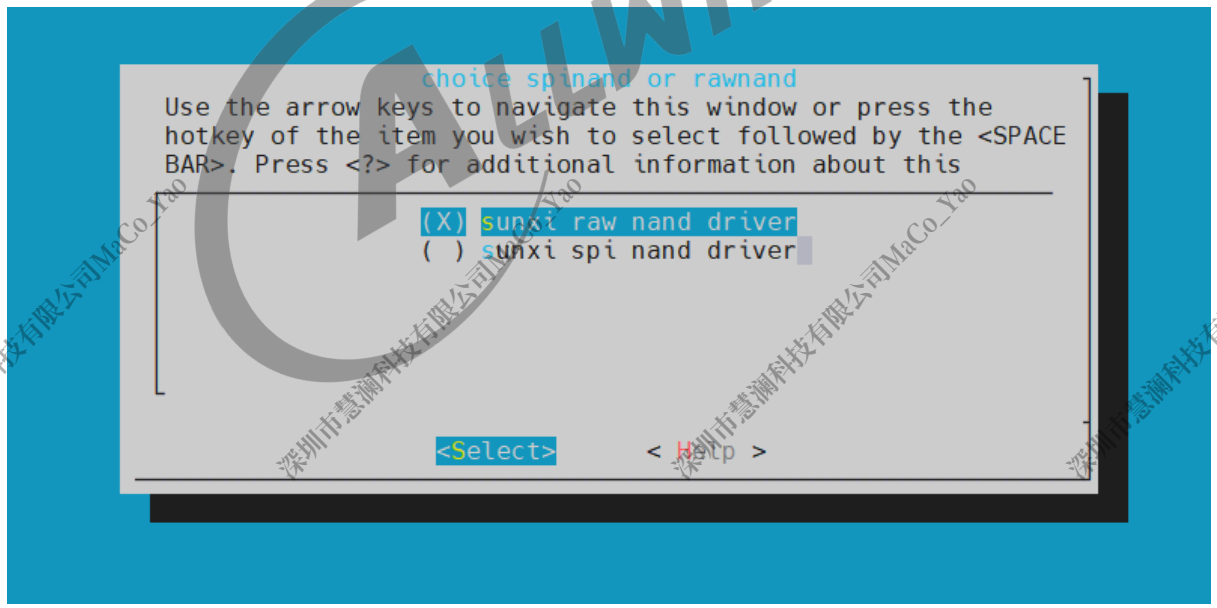


图 2-1: menuconfig 配置

```
pwroff-gpios功能选择：
Device Drivers->Device Drivers->Block devices->sunxi rawnand exteral module detect unclean
    poweroff
```

```
--- Block devices
<  >   Null test block driver
<*>   The sunxi nand driver
        choice spinand or rawnand (sunxi raw nand driver)  --->
[*]   sunxi rawnand exteral module detect unclean poweroff
<  >   Loopback device support
<  >   DRBD Distributed Replicated Block Device support
<  >   Network block device support
<  >   RAM block device support
<  >   Packet writing on CD/DVD media (DEPRECATED)
<  >   ATA over Ethernet support
<  >   Virtio block driver
<  >   Rados block device (RBD)
```

图 2-2

## 2.4 源码结构介绍

```
modules/nand/common1/
├── aw_nand_type.h
...
├── nfd
│   ├── nand_base.c
│   ├── nand_base.h
│   ├── nand_blk.c //与block层的对接
│   ├── nand_blk.h
│   ├── nand_boot.h
│   ├── nand_class.c //调试节点实现
│   ├── nand_class.h
│   ├── nand_dev.c  //nand设备管理
│   ├── nand_dev.h
│   ├── nand_lib.h
│   ├── nand_osal_for_linux.c  //nand版本管理
│   ├── nand_osal_for_linux.h
│   ├── nand_ota_burn.c  //OTA相关接口实现
│   ├── nand_ota_burn.h
│   ├── nand_panic.c  //panic时保存log到nand的接口实现
│   ├── nand_panic.h
...
├── nftl_v8.S  //nand nftl算法的汇编
├── phy2nftl.c
├── phy2nftl.h
└── phy-nand
   ...
    ├── rawnand  //raw nand的操作实现
    │   ├── controller  //NDFC控制器实现
    │   │   ├── Makefile
```

```
    │   │   ├── ndfc_base.c
    │   │   ├── ndfc_base.h
    │   │   ├── ndfc_internal.h
    │   │   ├── ndfc_ops.h
    │   │   ├── ndfc_timings.c
    │   │   ├── ndfc_timings.h
    │   │   ├── ndfc_v1px.c
    │   │   └── ndfc_v2px.c
    │   ├── Makefile
    │   ├── rawnand_base.c
    │   ├── rawnand_base.h
    │   ├── rawnand_boot0.c
    │   ├── rawnand_boot1.c
    │   ├── rawnand_boot.h
...
    ├── version.c
    └── version.h
```

## 2.5 驱动框架介绍

无

# 3 模块接口说明

## 3.1 NAND_ReadBoot1()

- 作用：读取 uboot 并返回到用户空间
- 参数：
  - 参数 1: 长度
  - 参数 2: 内存地址
- 返回：
  - 非零: 失败
  - 零: 成功

**⚠ 注意**

**该接口是通过 nand_ioctl 由用户层调用，实现读取 uboot**

## 3.2 NAND_ReadBoot0()

- 作用：读取 boot0 并返回到用户空间
- 参数：
- 参数 1: 长度
- 参数 2: 内存地址
- 返回：
- 非零: 失败
- 零: 成功

**⚠ 注意**

**该接口是通过 nand_ioctl 由用户层调用，实现读取 boot0**

## 3.3 NAND_BurnBoot0()

- 作用：更新 boot0

- 参数：
- 参数 1: 长度
- 参数 2: 内存地址
- 返回：
- 非零: 失败
- 零: 成功

**⚠ 注意**

**该接口是通过 nand_ioctl 由用户层调用，实现更新 boot0**

## 3.4 NAND_BurnBoot1()

- 作用：更新 uboot
- 参数：
- 参数 1: 长度
- 参数 2: 内存地址
- 返回：
- 非零: 失败
- 零: 成功

**⚠ 注意**

**该接口是通过 nand_ioctl 由用户层调用，实现更新 uboot**

## 3.5 nand_secure_storage_read()

- 作用：读取 secure storage 内容
- 参数：
- 参数 1:key 的条目
- 参数 2: 内存地址
- 参数 3: 长度
- 返回：
- 非零: 失败
- 零: 成功

**⚠ 注意**

**该接口是通过 nand_ioctl 由用户层调用，实现读取 secure storage**

## 3.6 nand_secure_storage_write()

- 作用：更新 uboot
- 参数：
- 参数 1:key 的条目
- 参数 2: 内存地址
- 参数 3: 长度
- 返回：
- 非零: 失败
- 零: 成功

**⚠ 注意**

**该接口是通过 nand_ioctl 由用户层调用，实现更新实现对 secure storage**

## 3.7 shutdown_flush_write_cache()

- 作用：刷 nand 驱动 cache 数据到 nand flash 中，保证数据一致性
- 参数：
- 参数 1: 无
- 返回：
- 零: 成功

**⚠ 注意**

**该接口是通过 shutdown 关机流程调用**

# 4 模块使用范例

无

# 5 FAQ

## 5.1 调试方法

### 5.1.1 调试工具

无

### 5.1.2 调试节点

#### 5.1.2.1 性能相关

- /sys/block/nand0 /queue/scheduler IO 调度器

目前 5.4 支持 mq-deadline kyber mq-bfq none，在复杂场景，用户交付频繁场景，建议适用 mq-bfq，其他场景使用 mq-deadline；

- /sys/block/nand0/queue/read_ahead_kb

最大预读数量，单位 KB，系统默认 128K，在进行大文件的读操作过程中可以适时调整该值，提升连续读操作性能

- /sys/block/nand0/queue/max_hw_sectors_kb

硬件最大的处理能力，建议用户可以利用这个最大带宽传输，提高读写性能，减少软件耗时

#### 5.1.2.2 NAND debug 相关

- nand_debug 节点

/sys/devices/platform/soc@2900000/4011000.nand0/nand_debug

“‘ eg: echo flush > nand_debug 主动刷 cache flush cache

eg: echo engcall > nand_debug 提供应用接口主动垃圾回收，如提示所示支持 128M 的小容量 flash gcc all invalid page count equal or large than page_per_blk / 2 for samll slc nand(128M)

eg: echo gcall xxx > nand_debug gcc all block of invalid page count equal or large than xxx

eg: echo gcone xxx > nand_debug gcc one block of invalid page count equal or large than xxx

eg: echo priogc xxx_b xxx_p> nand_debug prioritize to gcc the xxx_p pages in xxx_b ,when next time gc

eg: echo test 1 > nand_debug 打开一些调试打印 open some debug gate

eg: echo showall > nand_debug 打印所有 block 信息，包括擦除次数，编号，无效页个数等 output the free block and invalid block information

eg: echo showinfo > nand_debug 在串口打印 zone 管理信息，如空闲 block 个数，坏块个数，gc 信息，nand 配置信息等 output the zone information,after echo test 1 > nand_debug

eg: echo smart > nand_debug 在串口打印 smart 的管理信息，包括总共 gc 的次数，gc 的页数，wl 总次数和发生读写的总量等 output the smart information,after echo test 1 > nand_debug “‘

- nand_gcall

```
nand_gcone explain: use it can gc all block,   which invalid page count large than the input value;
   usage: echo xxx > nand_gcall
```

- nand_gcone

```
nand_gcone explain: use it can gc one block,   which invalid page count large than the input value;
   usage: echo xxx > nand_gcone
```

- nand_badblock cat 该值显示坏块个数
- sunxi_ndfc_reg cat 该值显示 NDFC 寄存器
- sunxi_nftl_version cat nand 驱动版本号
- sunxi_nand_pio_reg cat nand 引脚相关的寄存器

# 5.2 常见问题

## 5.2.1 NAND 驱动预留空间

驱动占用空间 ≈flash 总容量 *10%。这个 10% 的比例只是个粗略的估算，会随着 flash 的规格、贴片数以及坏块数而变化，下面会另有说明。

举例来说明使用标称 8GB 的 flash 容量占用情况:
各分区容量如下 (在 sys_partition.fex 文件中查看):
boot-resource：32MB
env：16MB
boot：16MB
system：768MB
data：1024MB（设置–》存储–》内部存储空间–》总容量）
misc：16MB
recovery：32MB
cache：512MB
databk：128MB
UDISK：剩余容量

(1)nand 驱动占用的容量 ≈8G*10%≈819.2M
(2) 其它分区占用的总容量 =（32+16+16+768+1024+16+32+512+128）MB=2544MB
(3)UDISK 分区的容量 ≈8G - 819.2M - 2544MB = 4828.8M

如果 UDISK 分区与实际容量差异较大，可检查 flash 是否有较多坏块，上面的算法中忽略了坏块的计算，坏块是要被从实际容量中抠除的。

## 5.2.2 影响剩余空间的因素

以上针对 nand flash 分区的计算都是基于 flash 一致性比较好的前提，如果 flash 本身坏块较多，再加上贴片数以及 flash 规格的影响，会造成不同 flash 之间剩余空间有很大的差异。
(1) 坏块
Nand flash 在出厂时都会有一定比例的坏块，在使用过程中也会产生坏块。坏块是不会再被使用的，坏块越多，可用的空间就会越少。
(2)Inter-leave 操作 (贴片数，一片 flash 包含多个 die)
如果板子上贴有多片 flash 或者单个 flash 中有多个 die 存在，nand 驱动则会启用 Inter-leave 操作。
Inter-leave 会将不同 bank 或者不同 chip 中的 block 捆绑使用，也就是说，产生了一个坏块，就会被放大成 2 个，nand 驱动预留一块，
也会被放大成 2 个，这样如果坏块很多，差异就会很多，而一般的预留块造成的差异则不会很明显，200M 以内属正常。

另外，Inter-leave 操作会导致超级块变大为原来的 2 倍，导致保留空间也变大为原来的 2 倍。

(3) 是否支持 two-pane

有些 flash 支持 two-plane，如果 nand 驱动又默认开启此操作的话，也会影响剩余容量。

two-plane 操作会将相同 bank 中的不同 plane 里的 block 捆绑一起使用，也就是说如果产生一个坏块，也会被放大成 2 个，

nand 驱动预留一个块，也会被放大成 2 个，这样如果坏块很多，差异就会很多，而一般的预留块造成的差异则不会很明显，

200M 以内属正常。

另外，twoplane 操作会导致超级块变大为原来的 2 倍，导致保留空间也成为原来的 2 倍。

(4)flash 本身 block 大小差异很大

显然 block 越大，坏块越多，不能使用的空间也就越多，剩余空间就相应减少了很多。当然也会影响预留空间产生的容量损耗，

这个影响是比较小的。

Two-plane 和 inter-leave 操作可以同时使用，虽然对坏块的容量损耗有放大作用，但对 flash 读写性能有显著的提升，而且一般的 flash 出厂坏块较少，这两个操作的影响也就微不足道了。

注意：nand 驱动是一般默认开启 two-pane 操作的，也就是说只要 flash 支持 two-plane, 就会起作用，这也就造成了支持 two-plane 的 flash 在这两个平台上的剩余空间会有差异。

举例来说明上述因素对容量的影响情况：将 kernel 的 loglevel 设置为最高等级，通过串口打印的 kernel log 就能看到上面所说的各个因素的支持情况以及坏块情况，下面举例说明：

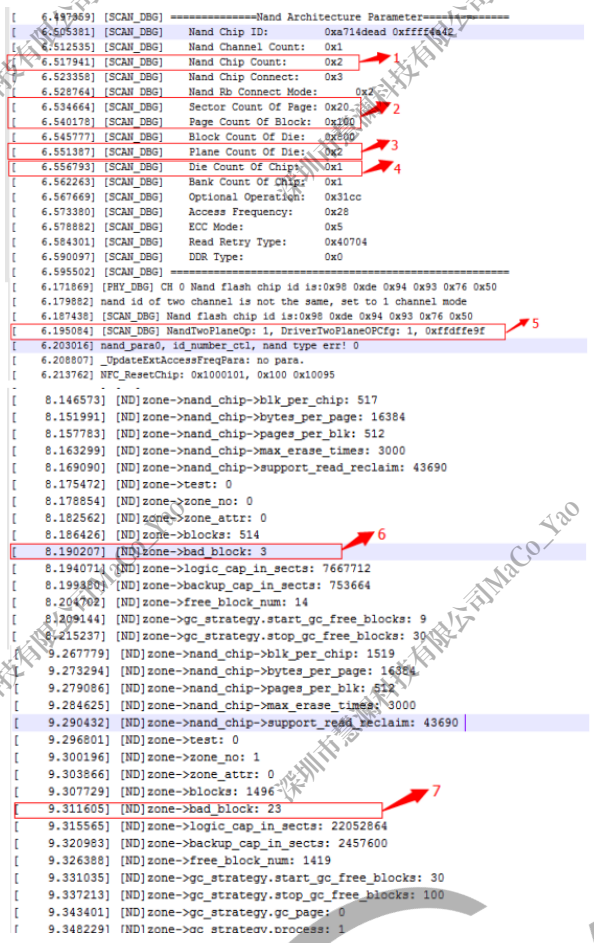


图 5-1: RAW 配置图

标注 3 说明了此片子是支持 two-plane 操作的，而标注 5 则说明驱动中打开了 two-plane 操作；通过标注 2 可以计算出块的大小，此例中 block_size=secttor_size * sectors * pages = 0x200 * 0x20 * 0x100 = 4M；标注 1 说明了板子上贴了几个片子，而标注 4 则说明每个片子中包含几个 die，如果它们中任何一个的值大于 2，就会开启 inter-leave 操作；标注 6 和标注 7 指示了坏块的数目，两者之和即是坏块总数，此例中坏块总数 =23+3=26 个。通过以上的说明可知，此案例支持 inter-leave 操作，坏块损耗翻倍，支持 two-plane 操作，坏块损耗再次翻倍，因此总的坏块损耗 = 坏块数 * block_size *2* 2 = 4M * 26 * 2 * 2 = 416M

## 5.2.3 提高可用容量的方法

### 5.2.3.1 分区分配

从第二节的计算方法中可以看出，为了提高用户可使用的容量，除了使用质量好的 flash 外，最重要的就是合理配置各个逻辑分区的大小；其实在公版中给出的方案中，为了让方案商改动较小快速实现方案，各个分区大小留有比较大的阈值，如果想提高用户可用容量，可以参考如下：

1. system 分区预留了 768MB，假如实际 system 分区编译出来的大小为 454M，配置成标准

的 512MB 是足够的，这样可以省下

768MB-512MB=256MB

2. cache 分区大小预留了 512MB，该分区主要用于 recovery 或者 OTA 升级，其容量最好大于完整包的大小；

   对无需过 cts 的平台来讲，使用 recovery 及 ota 的机会基本没有，用于升级完整包的机会更没有，一般最多用于升级差分包，预留 128MB 足够，
   这样可以省下 512MB-128MB=384MB

3. databk 分区预留了 128MB，该分区主要用于固件修改工具克隆功能使用的，如果用不到该功能，配置 16MB 即可，

   可以省下 128MB-16MB=112MB

   上述三种分区的修改，一定要根据实际情况处理，以免对后续使用产生影响。其他分区占用空间较小，且一般不要随意改动

总和以上，可以多分出 (256MB+384MB+112MB)*5/4=940MB*，*940MB*6/7=806MB，即用户可用空间可以多分出 806MB 空间具体分区大小配置需要根据自己方案的实际情况及功能需求来定义。

#### 5.2.3.2 NAND 配置

| 调整方案 | 影响 |
| --- | --- |
| nand0_capacity_level=0x1 | 1. 每个分区的保留比例由十分之一改为十三分之一。2. 重负载下随机写速度会降低。 |
| nand0_id_number_ctl = 0x2<br>nand0_p1 = 0xeeeeee | 1. 关闭 twoplane 操作，从而减少超级块的大小，使保留容量降低约为原来的二分之一。2. 降低顺序写速度。 |

Note: 客户需要折中性能与容量的关系，以免造成不必要的影响。

### 5.2.4 其他

可参考《NAND 量产问题 _ 排查指南》

**著作权声明**



本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

**商标声明**


、 
、全志科技、  （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

**免责声明**

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。