



# **Linux DMAC**

## **开发指南**

版本号: 2.2  
发布日期: 2020.04.15

## 版本历史

版本号	日期	制/修订人	内容描述
1.1	2020.06.29	AWA1440	1. 初版
2.0	2020.11.19	AWA1527	1. for linux-5.4
2.1	2021.04.08	XAA0190	1. 添加 linux-5.4 配置信息 2. 添加 linux-5.4 device tree 源码结构关系
2.2	2020.04.15	XAA0190	1. 修改格式



# 目 录

<b>1 概述</b>	<b>1</b>
1.1 编写目的 . . . . .	1
1.2 适用范围 . . . . .	1
1.3 相关人员 . . . . .	1
<b>2 DMA Engine 框架</b>	<b>2</b>
2.1 基本概述 . . . . .	2
2.1.1 术语约定 . . . . .	2
2.1.2 功能简介 . . . . .	2
2.2 基本结构 . . . . .	3
2.3 源码结构 . . . . .	3
2.4 模块配置 . . . . .	4
2.4.1 kernel menuconfig 配置 . . . . .	4
2.4.2 device tree 源码结构和路径 . . . . .	6
2.4.3 device tree 对 dma 控制器的通用配置 . . . . .	7
2.4.4 device tree 对 dma 申请者的配置 . . . . .	7
2.5 模式 . . . . .	7
2.5.1 内存拷贝 . . . . .	7
2.5.2 散列表 . . . . .	8
2.5.3 循环缓存 . . . . .	8
<b>3 模块接口说明</b>	<b>10</b>
3.1 dma_request_channel . . . . .	10
3.2 dma_request_chan . . . . .	10
3.3 dma_release_channel . . . . .	11
3.4 dmaengine_slave_config . . . . .	11
3.5 dmaengine_prep_slave_sg . . . . .	12
3.6 dmaengine_prep_dma_cyclic . . . . .	13
3.7 dmaengine_submit . . . . .	13
3.8 dma_async_issue_pending . . . . .	14
3.9 dmaengine_terminate_all . . . . .	14
3.10 dmaengine_pause . . . . .	14
3.11 dmaengine_resume . . . . .	15
3.12 dmaengine_tx_status . . . . .	15
<b>4 DMA Engine 使用流程</b>	<b>16</b>
4.1 基本流程 . . . . .	16
4.2 注意事项 . . . . .	16
<b>5 使用范例</b>	<b>17</b>
5.1 范例 . . . . .	17
<b>6 FAQ</b>	<b>19</b>

6.1 dma debug 宏 . . . . .	19
6.2 常见问题调试方法 . . . . .	21
6.3 利用 sunxi_dump 读写相应寄存器 . . . . .	21



## 插 图

2-1 DMA Engine 框架图 . . . . .	3
2-2 内核 menuconfig 根菜单 . . . . .	4
2-3 内核 menuconfig 根菜单 . . . . .	5
2-4 linux-4.9 内核 menuconfig dma drivers 菜单 . . . . .	5
2-5 linux-5.4 内核 menuconfig dma drivers 菜单 . . . . .	6
2-6 DMA Engine 内存拷贝示意图 . . . . .	7
2-7 DMA Engine 散列拷贝示意图 (slave 与 master) . . . . .	8
2-8 DMA Engine 散列拷贝示意图 (master 与 master) . . . . .	8
2-9 DMA Engine 循环拷贝示意图 . . . . .	9
4-1 DMA Engine 使用流程 . . . . .	16
6-1 内核 menuconfig 根菜单 . . . . .	19
6-2 内核 menuconfig 根菜单 . . . . .	20
6-3 内核 menuconfig 根菜单 . . . . .	20



# 1 概述

## 1.1 编写目的

介绍 DMA Engine 模块及其接口使用方法：

1. dma driver framework
2. API 介绍
3. 使用范例及注意事项

## 1.2 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
全志所有产品	Linux-4.9	sunxi-dma.c
全志所有产品	Linux-5.4	sun6i-dma.c

## 1.3 相关人员

- DMA 模块使用者
- 驱动模块负责人

## 2 DMA Engine 框架

### 2.1 基本概述

DMA Engine 是 linux 内核 dma 驱动框架，针对 DMA 驱动的混乱局面内核社区提出了一个全新的框架驱动，目标在统一 dma API 让各个模块使用 DMA 时不用关心硬件细节，同时代码复用提高。并且实现异步的数据传输，降低机器负载。

#### 2.1.1 术语约定

表 2-1: DMA 模块相关术语介绍

术语	解释说明
SUNXI	Allwinner 一系列 SOC 硬件平台
DMA	Direct Memory Access(直接内存存取)
Channel	DMA 通道
Slave	从通道，一般指设备通道
Master	主通道，一般指内存

#### 2.1.2 功能简介

DMA Engine 向使用者提供统一的接口，不同的模式下使用不同的 DMA 接口，降低使用者过多对硬件接口的关注。

## 2.2 基本结构

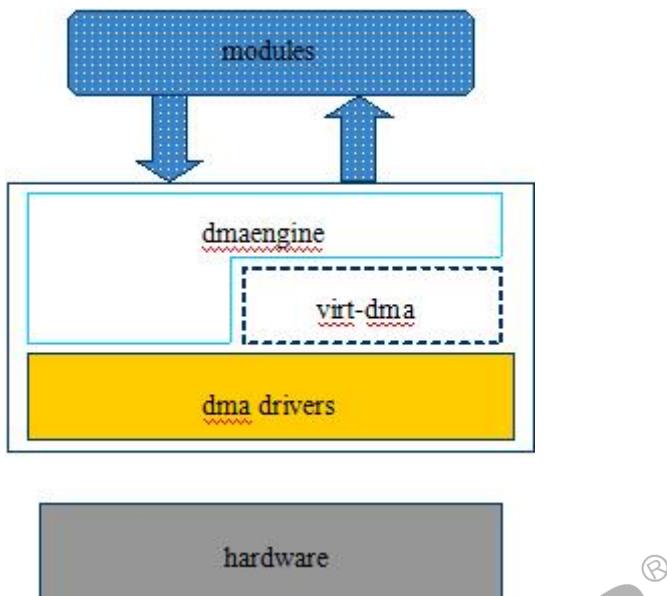


图 2-1: DMA Engine 框架图

## 2.3 源码结构

```
1 linux4.9
2 |
3 |--- drivers
4 |   `-- dma
5 |     |-- Kconfig
6 |     |-- Makefile
7 |     |-- dmaengine.c
8 |     |-- dmaengine.h
9 |     |-- of-dma.c
10 |    |-- virt-dma.c
11 |    |-- virt-dma.h
12 |    `-- sunxi-dma.c
13 |
14 `-- include
15   `-- linux
16     |-- sunxi
17     |   `---dma-sun*.h
18     `-- dma
19       `-- sunxi-dma.h
20
21 linux5.4
22 |
23 `-- drivers
24   `-- dma
25     |-- Kconfig
26     |-- Makefile
27     |-- dmaengine.c
28     |-- dmaengine.h
```

```
29      |-- of-dma.c
30      |-- virt-dma.c
31      |-- virt-dma.h
32      `-- sun6i-dma.c
```

## 2.4 模块配置

### 2.4.1 kernel menuconfig 配置

在命令行中进入 linux 目录，执行 make ARCH=arm64 menuconfig(32 位系统为 make ARCH=arm menuconfig) 进入配置主界面 (Linux-5.4 内核版本在 longan 目录下执行：./build.sh menuconfig, 在最后的配置中选择 Allwinner A31 SoCs DMA support)，并按以下步骤操作。

首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

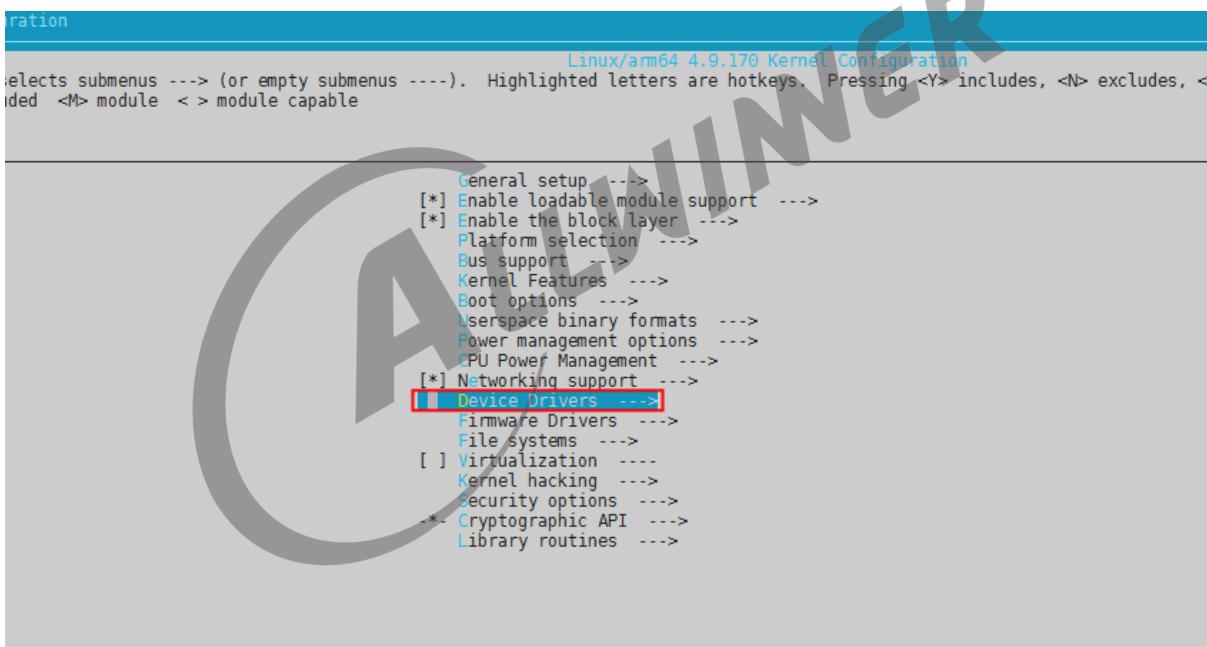


图 2-2: 内核 menuconfig 根菜单

选择 DMA Engine support, 进入下级配置，如下图所示：

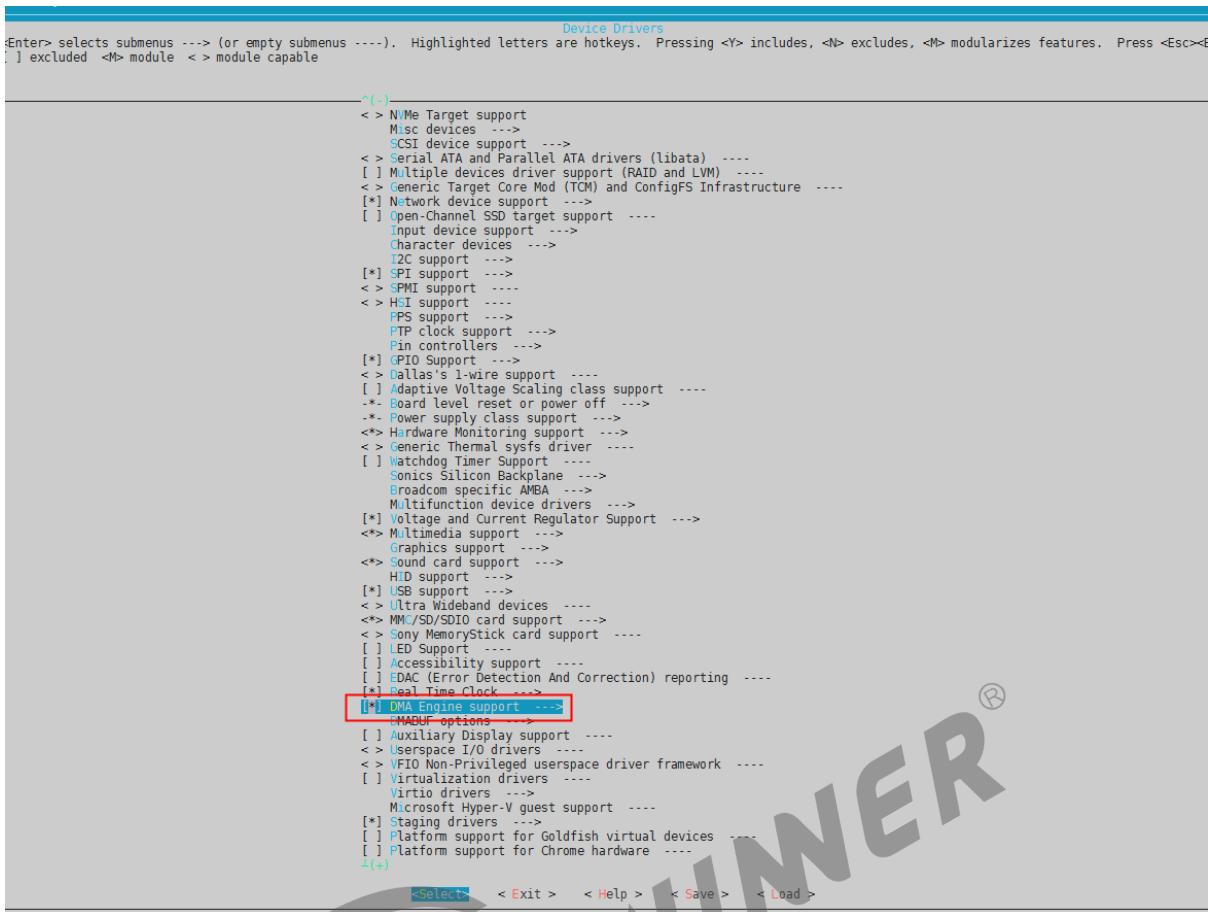


图 2-3: 内核 menuconfig 根菜单

linux-4.9 选择 Sunxi SOC DMA support 和 Support sunxi SOC DMA to access 4G address，如下图所示：

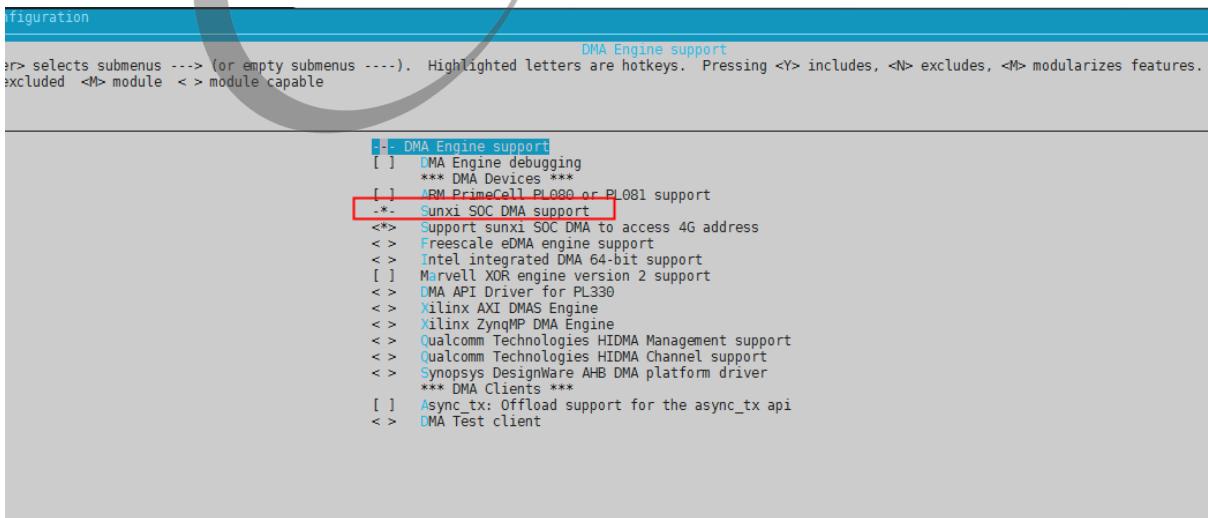


图 2-4: linux-4.9 内核 menuconfig dma drivers 菜单

linux-5.4 选择 Allwinner A31 SoCs DMA support，如下图所示：

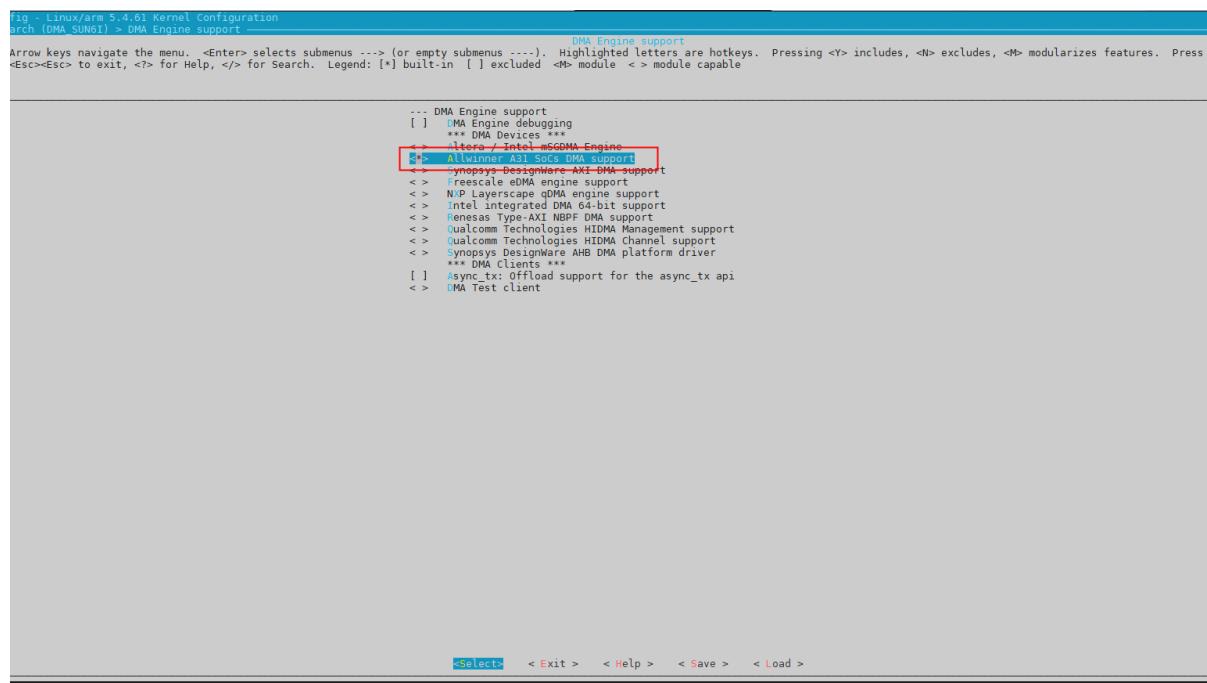


图 2-5: linux-5.4 内核 menuconfig dma drivers 菜单

## 2.4.2 device tree 源码结构和路径

- 设备树文件的配置是该 SOC 所有方案的通用配置，对于 ARM64 CPU 而言，设备树的路径为：kernel/{KERNEL\_VERSION}/arch/arm64/boot/dts/sunxi/sun\*.dtsi。
- 设备树文件的配置是该 SOC 所有方案的通用配置，对于 ARM32 CPU 而言，设备树的路径为：kernel/{KERNEL\_VERSION}/arch/arm/boot/dts/sun\*.dtsi。
- 板级设备树 (board.dts) 路径：/device/config/chips/{IC}/configs/{BOARD}/board.dts

linux4.9 device tree 的源码结构关系如下：

```

1 board.dts
2   |-----sun*.dtsi
3     |-----sun*-pinctrl.dtsi
4     |-----sun*-clk.dts

```

linux5.4 device tree 的源码结构关系如下：

```

1 board.dts
2   |-----sun*.dtsi

```

## 2.4.3 device tree 对 dma 控制器的通用配置

在 sun\*.dtsi 文件中，配置了该 SoC 的 dma 控制器的通用配置信息，一般不建议修改，由 dma 驱动维护者维护。

```
1  dma0:dma-controller@0x3002000 {  
2      compatible = "allwinner,sun50i-dma";          //兼容属性，用于驱动和设备绑定  
3      reg = <0x0 0x03002000 0x0 0x1000>;        //寄存器地址0x03002000和范围0x1000  
4      interrupts = <GIC_SPI 42 IRQ_TYPE_LEVEL_HIGH>; //dma控制器对应的gic硬中断号和触发类型  
5      clocks = <&clk_dma>;                      //dma使用的时钟，linux4.9配置在sun*-clk.dtsi中,linux5.4配置  
6      在sun*.dtsi中  
7      #dma-cells = <1>;                      //用于通过dts配置dma，目前没有使用  
};
```

## 2.4.4 device tree 对 dma 申请者的配置

在 sun\*.dtsi 文件中，配置了 SoC dma 控制器的申请者信息。

```
1  spi0: spi@5010000 {  
2      .....  
3      dmas = <&dma 22>, <&dma 22>;           //dma 通道号，参考dma spec  
4      dma-names = "tx", "rx";                  //dma 通道名字，与驱动对应  
5      .....  
6  };
```

## 2.5 模式

### 2.5.1 内存拷贝

纯粹的内存拷贝，即从指定的源地址拷贝到指定的目的地址。传输完毕会发生一个中断，并调用回调函数。

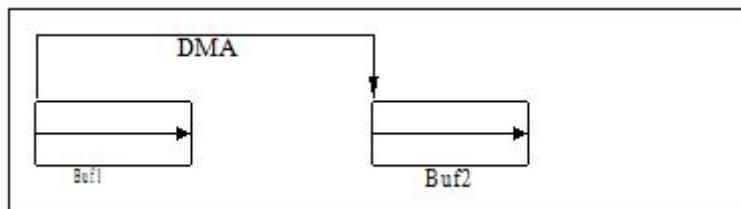


图 2-6: DMA Engine 内存拷贝示意图

## 2.5.2 散列表

散列模式是把不连续的内存块直接传输到指定的目的地址。当传输完毕会发生一个中断，并调用回调函数。

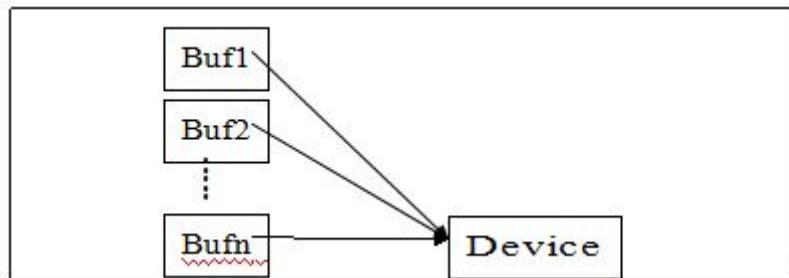


图 2-7: DMA Engine 散列拷贝示意图 (slave 与 master)

上述的散列拷贝操作是针对于 Slave 设备而言的，它支持的是 Slave 与 Master 之间的拷贝，还有另一散列拷贝是专门对内存进行操作的，即 Master 与 Master 之间进行操作，具体形式图如下：

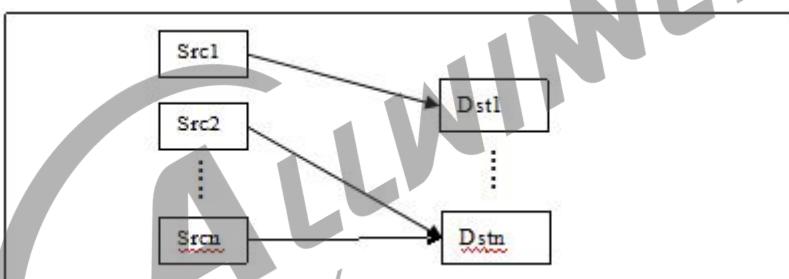


图 2-8: DMA Engine 散列拷贝示意图 (master 与 master)

## 2.5.3 循环缓存

循环模式就是把一块 Ring buffer 切成若干片，周而复始的传输，每传完一个片会发生一个中断，同时调用回调函数。

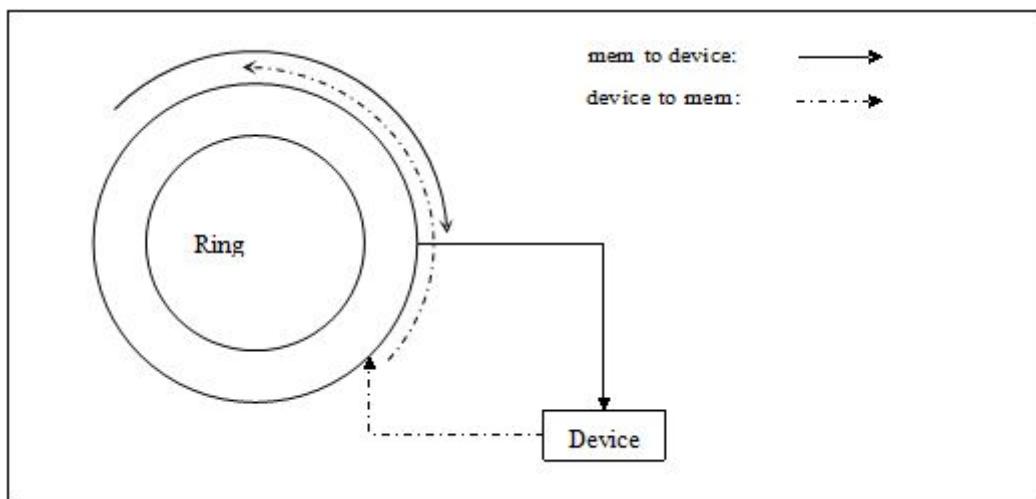


图 2-9: DMA Engine 循环拷贝示意图



## 3 模块接口说明

### 3.1 dma\_request\_channel

- 原型: `struct dma_chan *dma_request_channel(const dma_cap_mask_t *mask, dma_filter_fn fn, void *fn_param)`
- 作用: 申请一个可用通道, 返回 dma 通道操作句柄 (在 linux-5.4 上请使用 `dma_request_chan`)。
- 参数:
  - mask: 所有申请的传输类型的掩码。
  - fn: DMA 驱动私有的过滤函数, 可以为 NULL。
  - fn\_param: DMA 驱动私有的过滤函数, 传入的私有参数, 可以为 NULL。
- 返回:
  - 成功, 返回 dma 通道操作句柄。
  - 失败, 返回 NULL

### 3.2 dma\_request\_chan

- 原型: `struct dma_chan *dma_request_chan(struct device *dev, const char *name)`
- 作用: 申请一个可用通道, 返回 dma 通道操作句柄。
- 参数:
  - dev: 指向 dma 申请者的指针。
  - name: 通道名字, 与设备树的 `dma-names` 对应。
- 返回:
  - 成功, 返回 dma 通道操作句柄。
  - 失败, 返回 NULL

### 3.3 dma\_release\_channel

- 原型: void dma\_release\_channel(struct dma\_chan \*chan)
- 作用: 释放指定的 dma 通道。
- 参数:
  - chan: 指向要释放的 dma 通道句柄。
- 返回:
  - 无返回值

### 3.4 dmaengine\_slave\_config

- 原型: int dmaengine\_slave\_config(struct dma\_chan \*chan, struct dma\_slave\_config \*config)
- 作用: 配置 dma 通道的 slave 信息。
- 参数:
  - chan: 指向要操作的 dma 通道句柄。
  - config: dma 通道 slave 的参数。
- 返回:
  - 成功, 返回 0。
  - 失败, 返回错误码。



**dma\_slave\_config** 结构说明如下:

```
1 struct dma_slave_config {  
2     enum dma_transfer_direction direction;  
3     dma_addr_t src_addr;  
4     dma_addr_t dst_addr;  
5     enum dma_slave_buswidth src_addr_width;  
6     enum dma_slave_buswidth dst_addr_width;  
7     u32 src_maxburst;  
8     u32 dst_maxburst;  
9     bool device_fc;`  
10    unsigned int slave_id;  
11};  
12  
13 direction: 传输方向, 取值MEM_TO_DEV DEV_TO_MEM MEM_TO_MEM DEV_TO_DEV  
14
```

```

15 src_addr: 源地址，必须是物理地址
16 dst_addr: 目的地址，必须是物理地址
17
18 src_addr_width: 源数据宽度，byte整数倍，取值1, 2, 4, 8
19
20 dst_addr_width: 目的数据宽度，取值同上
21
22 src_max_burst: 源突发长度，取值1, 4, 8
23
24 dst_max_burst: 目的突发长度，取值同上
25
26 slave_id: 从通道id号，此处用作DRQ的设置，使用sunxi_slave_id(d, s)宏设置，具体取值参照include/linux/
27 sunxi-dma.h和include/linux/dma/sunxi/dma-sun*.h里使用。

```

## 说明

传输描述符介绍：

```

1 struct dma_async_tx_descriptor {
2     dma_cookie_t cookie;
3     enum dma_ctrl_flags flags; /* not a 'long' to pack with cookie */ ®
4     dma_addr_t phys;
5     struct dma_chan *chan;
6     dma_cookie_t (*tx_submit)(struct dma_async_tx_descriptor *tx);
7     dma_async_tx_callback callback;
8     void *callback_param;
9 };
10
11 cookie: 本次传输的cookie，在此通道上唯一
12
13 tx_submit: 本次传输的提交执行函数
14
15 callback: 传输完成后的回调函数
16
17 callback_param: 回调函数的参数

```

### 3.5 dmaengine\_prep\_slave\_sg

- 原型：

```
struct dma_async_tx_descriptor *dmaengine_prep_slave_sg(struct dma_chan *chan, struct scatterlist *sgl, unsigned int sg_len, enum dma_transfer_direction dir, unsigned long flags, void *context)
```

- 作用：准备一次单包传输。
- 参数：
  - chan: 指向要操作的 dma 通道句柄。
  - sgl: 散列表地址，此散列表传输之前需要建立。

- sg\_len: 散列表内 buffer 的个数。
  - dma\_transfer\_direction dir: 传输方向，此处为 DMA\_MEM\_TO\_DEV, DMA\_DEV\_TO\_MEM。
  - flags: 传输标志。
- 返回：
    - 成功，返回一个传输描述符指针。
    - 失败，返回 NULL。

### 3.6 dmaengine\_prep\_dma\_cyclic

- 原型：

```
struct dma_async_tx_descriptor *dmaengine_prep_dma_cyclic(struct dma_chan *chan, dma_addr_t buf_addr  
, size_t buf_len, size_t period_len, enum dma_transfer_direction dir, unsigned long flags)
```

- 作用：准备一次环形 buffer 传输。
- 参数：
  - chan: 指向要操作的 dma 通道句柄。
  - buf\_addr: 目的地地址。
  - buf\_len: 环形 buffer 的长度。
  - period\_len: 每一小片 buffer 的长度。
  - dma\_transfer\_direction dir: 传输方向，此处为 DMA\_MEM\_TO\_DEV, DMA\_DEV\_TO\_MEM。
  - flags: 传输标志。
- 返回：
  - 成功，返回一个传输描述符指针。
  - 失败，返回 NULL。

### 3.7 dmaengine\_submit

- 原型：dma\_cookie\_t dmaengine\_submit(struct dma\_async\_tx\_descriptor \*desc)
- 作用：提交已经做好准备的传输。
- 参数：
  - desc: 指向要提交的传输描述符。

- 返回：
  - 成功，返回一个大于 0 的 cookie。
  - 失败，返回错误码。

## 3.8 dma\_async\_issue\_pending

- 原型：void dma\_async\_issue\_pending(struct dma\_chan \*chan)
- 作用：启动通道传输。
- 参数：
  - chan: 指向要使用的通道。
- 返回：
  - 无返回值。

## 3.9 dmaengine\_terminate\_all

- 原型：int dmaengine\_terminate\_all(struct dma\_chan \*chan)
- 作用：停止通道上的所有传输。
- 参数：
  - chan: 指向要终止的通道。
- 返回：
  - 成功，返回 0。
  - 失败，返回错误码。

### 警告

此功能会丢弃未开始的传输。

## 3.10 dmaengine\_pause

- 原型：int dmaengine\_pause(struct dma\_chan \*chan)
- 作用：暂停某通道的传输。
- 参数：
  - chan: 指向要暂停传输的通道
- 返回：

- 成功，返回 0。
- 失败，返回错误码。

### 3.11 dmaengine\_resume

- 原型：int dmaengine\_resume(struct dma\_chan \*chan)
- 作用：恢复某通道的传输。
- 参数：
  - chan: 指向要恢复传输的通道。
- 返回：
  - 成功，返回 0。
  - 失败，返回错误码。

### 3.12 dmaengine\_tx\_status

- 原型：enum dma\_status dmaengine\_tx\_status(struct dma\_chan \*chan, dma\_cookie\_t cookie, struct dma\_tx\_state \*state)
- 作用：查询某次提交的状态。
- 参数：
  - chan: 指向要查询传输状态的通道。
  - cookie: dmaengine\_submit 接口返回的 id。
  - state: 用于获取状态的变量地址。
- 返回：
  - DMA\_SUCCESS，表示传输成功完成。
  - DMA\_IN\_PROGRESS，表示提交尚未处理或处理中。
  - DMA\_PAUSE，表示传输已经暂停。
  - DMA\_ERROR，表示传输失败。

## 4 DMA Engine 使用流程

本章节主要是讲解 DMA Engine 的使用流程，以及注意事项

### 4.1 基本流程

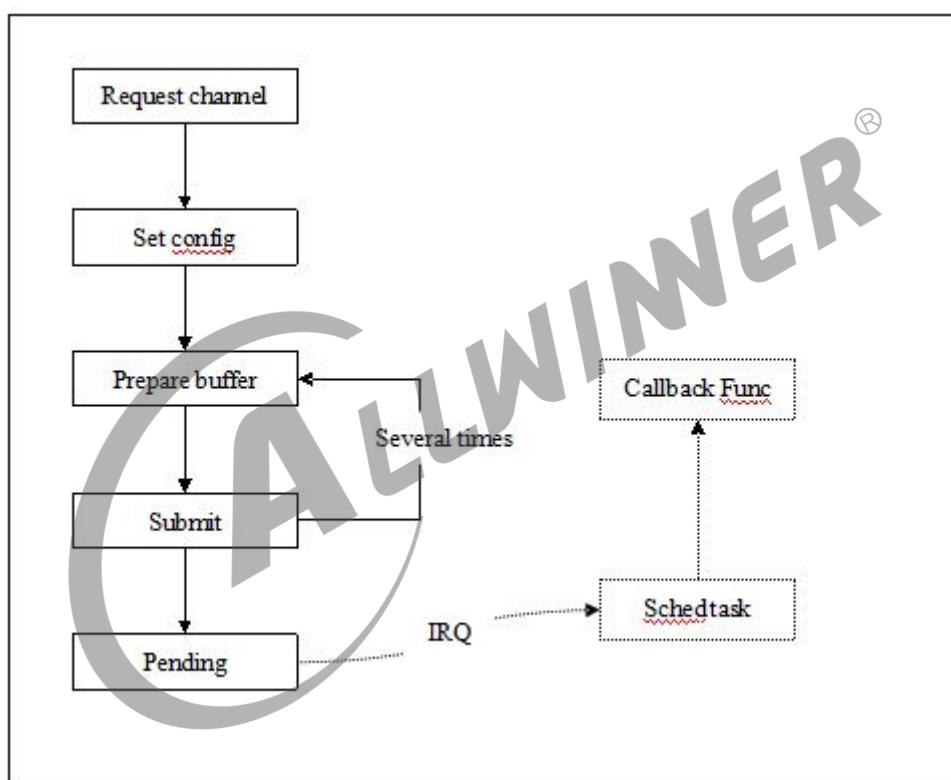


图 4-1: DMA Engine 使用流程

### 4.2 注意事项

- 回调函数里不允许休眠，以及调度
- 回调函数时间不宜过长
- Pending 并不是立即传输而是等待软中断的到来，cyclic 模式除外
- 对于 linux-4.9，在 dma\_slave\_config 中的 slave\_id 对于 devices 必须要指定

## 5 使用范例

```
1 struct dma_chan *chan;
2 dma_cap_mask_t mask;
3 dma_cookie_t cookie;
4 struct dma_slave_config config;
5 struct dma_tx_state state;
6 struct dma_async_tx_descriptor *tx = NULL;
7 void *src_buf;
8 dma_addr_t src_dma;
9
10 dma_cap_zero(mask);
11 dma_cap_set(DMA_SLAVE, mask);
12 dma_cap_set(DMA_CYCLIC, mask);
13
14 /* 申请一个可用通道 */
15 chan = dma_request_channel(dt->mask, NULL, NULL);
16 if (!chan){
17     return -EINVAL;
18 }
19
20 src_buf = kmalloc(1024*4, GFP_KERNEL);
21 if (!src_buf) {
22     dma_release_channel(chan);
23     return -EINVAL;
24 }
25
26 /* 映射地址用DMA访问 */
27 src_dma = dma_map_single(NULL, src_buf, 1024*4, DMA_TO_DEVICE);
28
29 config.direction = DMA_MEM_TO_DEV;
30 config.src_addr = src_dma;
31 config.dst_addr = 0x01c;
32 config.src_addr_width = DMA_SLAVE_BUSWIDTH_2_BYTES;
33 config.dst_addr_width = DMA_SLAVE_BUSWIDTH_2_BYTES;
34 config.src_maxburst = 1;
35 config.dst_maxburst = 1;
36 config.slave_id = sunxi_slave_id(DRQDST_AUDIO_CODEC, DRQSRC_SDRAM);
37
38 dmaengine_slave_config(chan, &config);
39
40 tx = dmaengine_prep_dma_cyclic(chan, scr_dma, 1024*4, 1024, DMA_MEM_TO_DEV,
41                               DMA_PREP_INTERRUPT | DMA_CTRL_ACK);
42
43 /* 设置回调函数 */
44 tx->callback = dma_callback;
45 tx->callback = NULL;
46
47 /* 提交及启动传输 */
```

```
48 cookie = dmaengine_submit(tx);  
49 dma_async_issue_pending(chan);
```



## 6 FAQ

### 6.1 dma debug 宏

在内核的 menuconfig 菜单项中使能该选项后，在 dma 传输时，会打印 dma 描述符，寄存器和其他一些 debug 信息，有助于我们进行 debug。该菜单配置项的打开方式如下：

在命令行中进入内核根目录 (kernel/linux-4.9)，执行 make ARCH=arm64(arm) menuconfig 进入配置主界面，并按以下步骤操作：首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

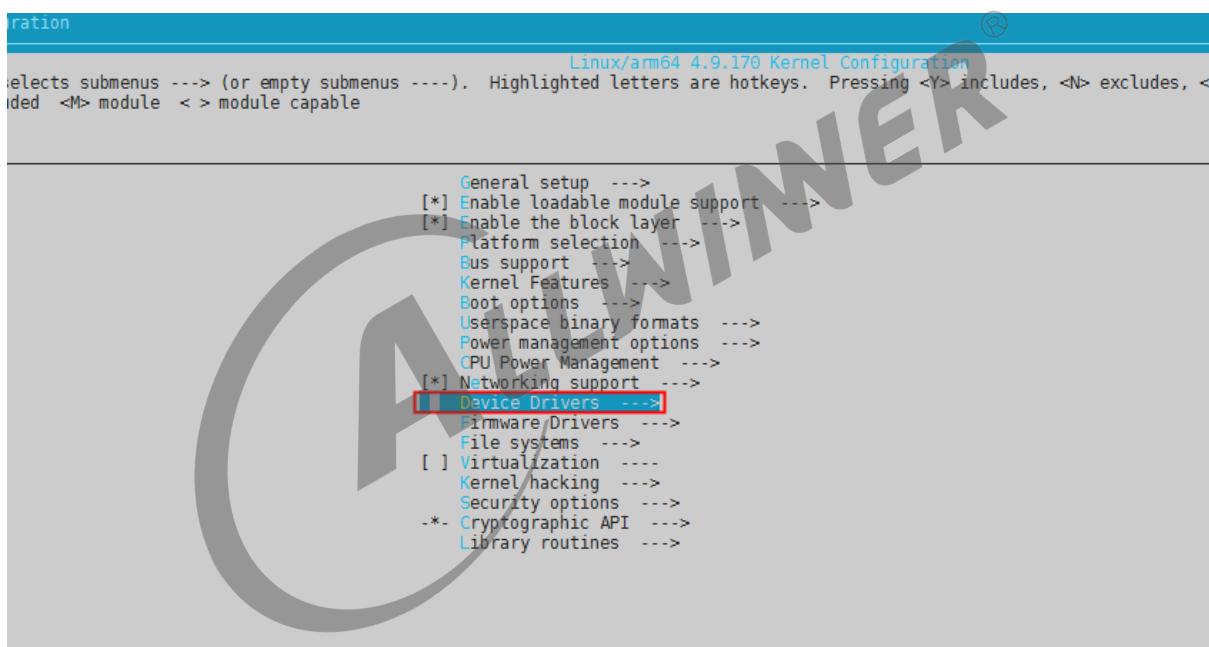


图 6-1: 内核 menuconfig 根菜单

选择 DMA Engine support, 进入下级配置，如下图所示：

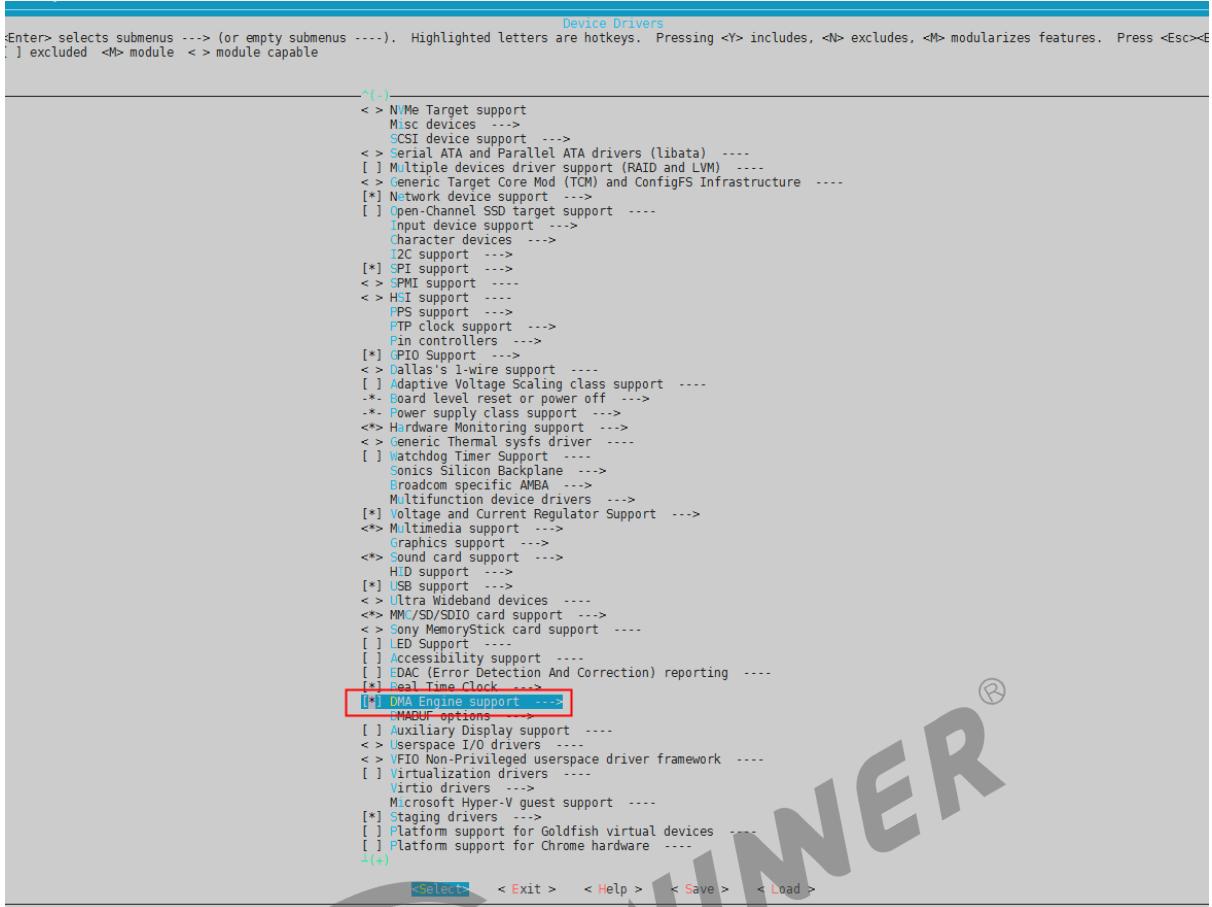


图 6-2: 内核 menuconfig 根菜单

选择 DMA Engine debugging, 如下图所示：

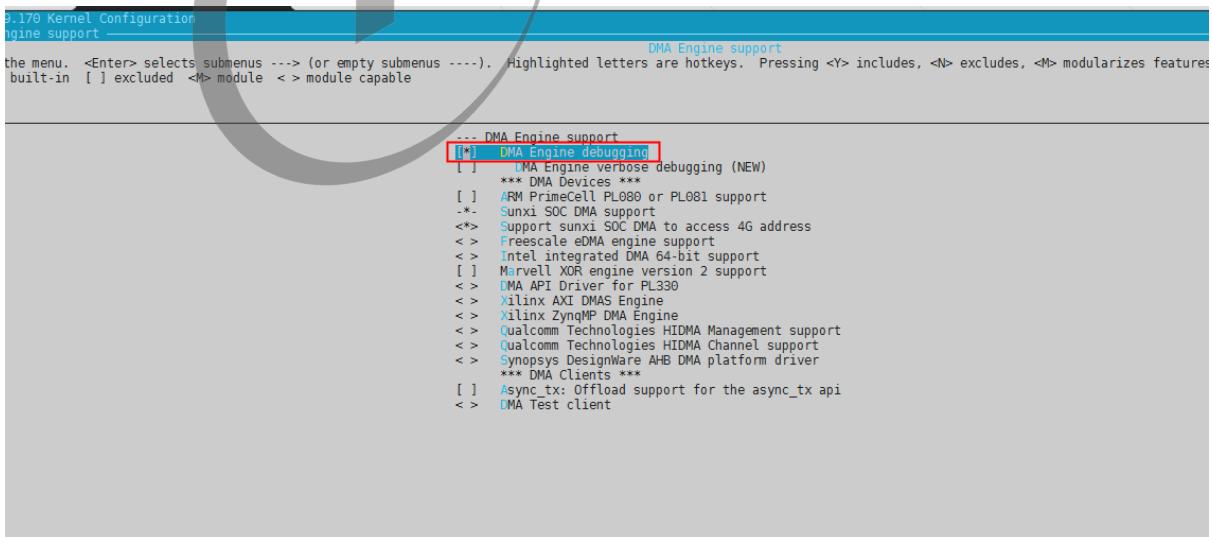


图 6-3: 内核 menuconfig 根菜单

把 CONFIG\_DMADEVICES\_DEBUG 这个配置打开后，在使用 dma 时，会有一些对应的打印调试信息，方便我们定位问题

## 6.2 常见问题调试方法

## 6.3 利用 sunxi\_dump 读写相应寄存器

```
1 cd /sys/class/sunxi_dump/
2 1.查看一个寄存器
3 echo 0x03002000 > dump ;cat dump
4
5 结果如下：
6 cupid-p1:/sys/class/sunxi_dump # echo 0x03002000 > dump ;cat dump
7 0x00000022
8
9 2.写值到寄存器上
10 echo 0x03002000 0x1 > write ;cat write
11
12 3.查看一片连续寄存器
13 echo 0x03002000,0x03002fff > dump;cat dump
14
15 结果如下：
16 cupid-p1:/sys/class/sunxi_dump # echo 0x03002000,0x03002fff > dump;cat dump
17
18 0x000000003002000: 0x00000022 0x00000000 0x00000000 0x00000000
19 0x000000003002010: 0x00000000 0x00000000 0x00000000 0x00000000
20 0x000000003002020: 0x000000ff 0x00000000 0x00000007 0x00000000
21 0x000000003002030: 0x00000000 0x00000000 0x00000000 0x00000000
22 0x000000003002040: 0x00030000 0x00000000 0x00000000 0x00000000
23 0x000000003002050: 0x00000000 0x00000000 0x00000000 0x00000000
24 0x000000003002060: 0x00000000 0x00000000 0x00000000 0x00000000
25 0x000000003002070: 0x00000000 0x00000000 0x00000000 0x00000000
26 0x000000003002080: 0x00000000 0x00000000 0x00000000 0x00000000
27 0x000000003002090: 0x00000000 0x00000000 0x00000000 0x00000000
28 0x0000000030020a0: 0x00000000 0x00000000 0x00000000 0x00000000
29 0x0000000030020b0: 0x00000000 0x00000000 0x00000000 0x00000000
30 0x0000000030020c0: 0x00000000 0x00000000 0x00000000 0x00000000
31 0x0000000030020d0: 0x00000000 0x00000000 0x00000000 0x00000000
32 0x0000000030020e0: 0x00000000 0x00000000 0x00000000 0x00000000
33 0x0000000030020f0: 0x00000000 0x00000000 0x00000000 0x00000000
34 0x000000003002100: 0x00000000 0x00000000 0xfc0000e0 0x83460240
35 0x000000003002110: 0xfc106500 0x05096020 0x00000b80 0x00010008
36 0x000000003002120: 0x00000000 0x00000000 0x0000000c 0xfc0000c0
37 0x000000003002130: 0x00000000 0x00000000 0x00000000 0x00000000
38 0x000000003002140: 0x00000000 0x00000000 0xfc0001e0 0x83430240
39 0x000000003002150: 0xfc506200 0x05097030 0x00000e80 0x00010008
40 0x000000003002160: 0x00000000 0x00000000 0x0000000c 0xfc0001c0
41 0x000000003002170: 0x00000000 0x00000000 0x00000000 0x00000000
42 0x000000003002180: 0x00000000 0x00000000 0x00000000 0x00000000
43 0x000000003002190: 0x00000000 0x00000000 0x00000000 0x00000000
44 0x0000000030021a0: 0x00000000 0x00000001 0x00000000 0x00000000
45 0x0000000030021b0: 0x00000000 0x00000000 0x00000000 0x00000000
46 0x0000000030021c0: 0x00000000 0x00000000 0x00000000 0x00000000
47 0x0000000030021d0: 0x00000000 0x00000000 0x00000000 0x00000000
48 0x0000000030021e0: 0x00000000 0x00000001 0x00000000 0x00000000
49 0x0000000030021f0: 0x00000000 0x00000000 0x00000000 0x00000000
50 0x000000003002200: 0x00000000 0x00000000 0x00000000 0x00000000
51 0x000000003002210: 0x00000000 0x00000000 0x00000000 0x00000000
52 0x000000003002220: 0x00000000 0x00000001 0x00000000 0x00000000
```

```
53 0x00000000003002230: 0x00000000 0x00000000 0x00000000 0x00000000
54 0x00000000003002240: 0x00000000 0x00000000 0x00000000 0x00000000
55 0x00000000003002250: 0x00000000 0x00000000 0x00000000 0x00000000
56 0x00000000003002260: 0x00000000 0x00000001 0x00000000 0x00000000
57 0x00000000003002270: 0x00000000 0x00000000 0x00000000 0x00000000
58 0x00000000003002280: 0x00000000 0x00000000 0x00000000 0x00000000
59 0x00000000003002290: 0x00000000 0x00000000 0x00000000 0x00000000
60 0x000000000030022a0: 0x00000000 0x00000001 0x00000000 0x00000000
61 0x000000000030022b0: 0x00000000 0x00000000 0x00000000 0x00000000
62 0x000000000030022c0: 0x00000000 0x00000000 0x00000000 0x00000000
63 0x000000000030022d0: 0x00000000 0x00000000 0x00000000 0x00000000
64 0x000000000030022e0: 0x00000000 0x00000001 0x00000000 0x00000000
65 0x000000000030022f0: 0x00000000 0x00000000 0x00000000 0x00000000
66 0x00000000003002300: 0x00000000 0x00000000 0x00000000 0x00000000
67 0x00000000003002310: 0x00000000 0x00000000 0x00000000 0x00000000
68 0x00000000003002320: 0x00000000 0x00000001 0x00000000 0x00000000
69 0x00000000003002330: 0x00000000 0x00000000 0x00000000 0x00000000
70 0x00000000003002340: 0x00000000 0x00000000 0x00000000 0x00000000
71 0x00000000003002350: 0x00000000 0x00000000 0x00000000 0x00000000
72 0x00000000003002360: 0x00000000 0x00000001 0x00000000 0x00000000
73 0x00000000003002370: 0x00000000 0x00000000 0x00000000 0x00000000
74 0x00000000003002380: 0x00000000 0x00000000 0x00000000 0x00000000
75 0x00000000003002390: 0x00000000 0x00000000 0x00000000 0x00000000
76 0x000000000030023a0: 0x00000000 0x00000001 0x00000000 0x00000000
77 0x000000000030023b0: 0x00000000 0x00000000 0x00000000 0x00000000
78 0x000000000030023c0: 0x00000000 0x00000000 0x00000000 0x00000000
79 0x000000000030023d0: 0x00000000 0x00000000 0x00000000 0x00000000
80 0x000000000030023e0: 0x00000000 0x00000001 0x00000000 0x00000000
```

通过上述方式，可以查看，从而发现问题所在。

## 著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

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。