Linux DMAC
开发指南

版本号: 1.12
发布日期: 2025.6.21

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.07.28	AWA0270	初始版本
1.1	2022.08.03	AWA0270	修改错误配合以及错误描述
1.2	2022.11.07	AWA0190	添加适用范围
1.3	2022.12.07	AWA0190	根据评审意见修改（修改 dtsi 配置等）
1.4	2023.02.15	AWA0190	添加 DMA 控制器等介绍
1.5	2023.03.16	AWA0190	修改 2.4 章节英文字体格式
1.6	2023.04.23	AWA0190	修改 1.2.2 章节文字描述
1.7	2023.04.28	AWA0190	1.8.3 章节添加多核公用 dma 控制器时注意事项
1.8	2024.03.13	XAA0228	优化 2.5 章节 config 配置和 dts 配置指导
1.9	2024.07.24	XAA0250	新增 NDMA 配置方法及说明.
1.1	2024.10.22	XAA0249	1. 修复部分图片不能显示的问题和标题问题
1.11	2025.2.25	XAA0249	1. 补充部分细节描述
1.12	2025.6.21	XAA0331	1. 增加 DMA 硬件隔离功能适配指南

目　录

插　图

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: 适用内核驱动列表

IP	驱动文件
DMA(SGDMA)	{SDK}/bsp/drivers/dma/sunxi-dma.c
NDMA	{SDK}/bsp/drivers/dma/sunxi-ndma.c

注意：若外发 SDK 中使用 BSP 独立驱动，均与对应的 SDK 内核版本相适用，不局限域上述已列的内核版本。

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

2.1.1 DMA 控制器

DMA 控制器主要实现传输将数据从一个地址空间复制到另一个地址空间，提供在外设和存储器之间或者存储器和存储器之间的高速数据传输。当 CPU 初始化这个传输动作，传输动作本身是由 DMA 控制器来实现和完成的。DMA 传输方式无需 CPU 直接控制传输，也没有中断处理方式那样保留现场和恢复现场过程，通过硬件为 RAM 和 IO 设备开辟一条直接传输数据的通道，使得 CPU 的效率大大提高。控制器提供了多个通道，每个通道都支持外设和 MEM 的 DMA 请求。

2.1.2 DMA Engine

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，同时提高代码复用性，降低 CPU 负载。

2.1.3 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

2.1.4 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

DMA 基本框架结构如下：module 为实际使用 DMA 的模块；dma drivers 为 DMA 底层驱动；dmaengine 为 DMA 内核框架；hardware 为 DMA 硬件；

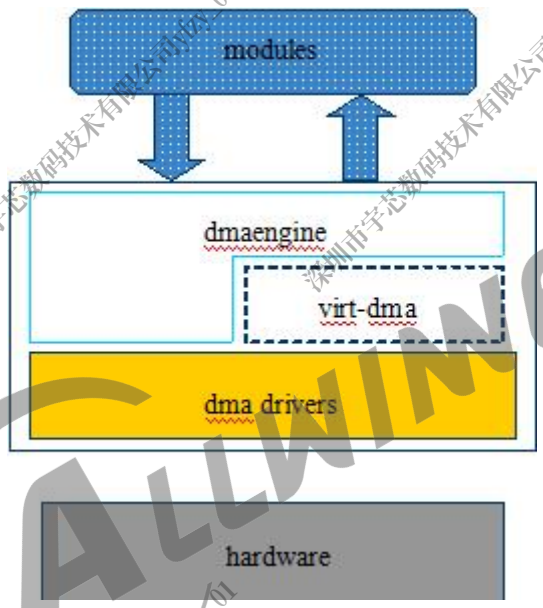

图 2-1: DMA Engine 框架图

2.3 源码结构

```
${SDK}
  ├──bsp/drivers/dma
        ├──── Kconfig
        ├──── Makefile
        └──── sunxi-dma.c
```

2.4 可编程数据宽度

DMA 控制器的通道可支持以不同数据宽度为单位进行一次搬运：1Byte，2Byte, 4Byte，8Byte；通过在如下数据结构体中 xx_width 参数进行设置，可传输给 DMA 控制器，进行不同数据宽度的搬运。DMA 控制器的通道可支持配置一次搬运的数据长度（以位宽为基础成倍数）：1，4，8；通

过在如下数据结构体中 xx_max_burst 参数进行设置，可传输给 DMA 控制器，进行一次搬运数据长度的配置。通常情况下需要设置 width×burst 与设备端 fifo trigger level 成倍数关系。若 trigger level 设置为 24，则 width×burst 需被 24 整除（如 12）或等于 24。

```
struct dma_slave_config {
    enum dma_transfer_direction direction;
    dma_addr_t src_addr;
    dma_addr_t dst_addr;
    enum dma_slave_buswidth src_addr_width;
    enum dma_slave_buswidth dst_addr_width;
    u32 src_maxburst;
    u32 dst_maxburst;
    bool device_fc;
    unsigned int slave_id;
};

src_addr_width:   源数据宽度，byte整数倍，取值1，2，4，8;在代码中对应DMA_SLAVE_BUSWIDTH_1_BYTE~
      DMA_SLAVE_BUSWIDTH_8_BYTE

dst_addr_width:   目的数据宽度，取值同上

src_max_burst:   源突发长度，取值1，4，8

dst_max_burst:   目的突发长度，取值同上
```

2.5 模块配置

2.5.1 kernel menuconfig 配置

1. 若板型支持系统 DMA，需要确保如下配置打开，方可正常使用系统 DMA：

```
CONFIG_AW_DMA=y
```

2. 若板型支持系统 NDMA，需要确保如下配置打开，方可正常使用系统 DMA：

```
CONFIG_AW_NDMA=y
```

说明：客户可根据实际需求，配置成 build-in 或者配置成 ko 均可；但是 Allwinner SDK 默认均已打开系统 DMA 配置，客户无需重复配置。

2.5.2 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，例如linux-5.10设备树配置目录：$SDK/bsp/configs/linux-5.10/${chip}.dtsi。

linux5.10 device tree的源码结构关系如下：

```
board.dts
|--------${chip}.dtsi
```

2.5.3 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. compatible = "allwinner,dma-v100";

无需更改配置项：此配置表示该 SOC 支持系统 DMA 的哪个版本驱动，客户无需更改；

2. reg = <0x0 0x03002000 0x0 0x1000>;

无需更改配置项：此配置表示该 SOC 系统 DMA 寄存器基地址和范围，客户无需更改，用于使用sunxi_dump调试时参考；

3. dma-channels = <8>;

可选更改配置项：此配置表示该 SOC 系统 DMA 支持的 DMA 通道数，客户视情况更改，但是 All-winner SDK 已配置正常，一般无需更改；

说明

当使用场景存在多核复用同一DMA控制器的情况时，若 DMA 通道总数为 16，则一般 Linux 端通道数仅配置为 8；另外 8 个通道在其他系统进行使用；此时上述配置中 dma 通道配置如下设置：dma-channels = <8>;

4. dma-requests = <52>;

无需更改配置项：此配置表示该 SOC 系统 DMA 支持的最大 DMA DRQ 号，客户无需更改；

说明：其他配置项均为常规参数配置，我们默认已配置正常，上述几项参数均为客户需要知晓的配置，若还有其他配置需要客户关注，会像上述参数一样针对性说明，并及时更新本文档。

2.5.4 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，以 spi0 驱动为例：

```
spi0: spi@5010000 {
  ......
  dmas = <&dma 22>, <&dma 22>;       //dma 通道号，参考dma spec，注意"&dma"部分需要根据模块所用dma控制器实际情况填写，常用"&dma"控制器或"&dma1"控制器
  dma-names = "tx", "rx";            //dma 通道名字,与驱动对应
```

```
    ......
};
```

2.5.5 DMA 通道隔离

2.5.5.1 DMA 通道隔离原理

DMA 通道隔离实现了每个 DMA 通道相关寄存器的访问权限限制，DMA 的每个通道可以分配给多个 user group 中的 1 个，每个 user group 可包含多个 user(CPU 核），若没有权限的 CPU 核访问了 DMA 通道的寄存器，不会产生中断。

2.5.5.2 DMA 通道隔离配置

在board.dts文件中对 DMA 通道进行配置，配置方法如下：

```
&amp_sys_rsc_mgr {
 status = "okay";

 ug1:user_group@1 {
  users = <RV_E907_CPU0>;
 };

 ug2:user_group@2 {
  users = <ARM_A7_CPU0>, <ARM_A7_CPU1>, <ARM_A7_CPU2>, <ARM_A7_CPU3>;
 };

 dma_resource_manager@0 { /* 第一个DMA控制器(dma) */
  controller = <&dma>;
  hw_isolation;
  hw_user_group_num = <2>;
  used_for_linux {
   channels = <0>, <1>, <2>, <3>,
      <4>, <5>, <6>, <7>,
      <8>, <9>, <10>, <11>,
      <12>, <13>, <14>, <15>;
    owner = <&ug2>;
  };

  /*
  used_for_rv {
   channels = <>;
   owner = <&ug1>;
  };
  */
 };

 dma_resource_manager@1 { /*  第二个DMA控制器(dma1) */
  controller = <&dma1>;
  hw_isolation;
  hw_user_group_num = <2>;
  /*
```

```
  used_for_linux {
    channels = <>;
    owner = <&ug2>;
  };
  */

  used_for_rv {
    channels = <0>, <1>, <2>, <3>,
        <4>, <5>, <6>, <7>;
    owner = <&ug1>;
  };
 };
};
```

2.5.5.3 配置解释

```
hw_isolation;
hw_user_group_num = <2>;
```

可选更改配置项：其中 hw_isolation 表示启动通道隔离，hw_user_group_num = <2> 表示通道隔离支持两个用户组，若想关闭通道隔离功能，删除掉这两配置即可。

```
used_for_linux {
  channels = <0>, <1>, <2>, <3>, <4>, <5>, <6>, <7>,<8>, <9>, <10>, <11>, <12>, <13>, <14>, <15>;
  owner = <&ug2>;
}
```

可选更改配置项:used_for_linux 表示该配置在 linux 端生效，channels 表示通道分配，上述配置是将 dma 通道 0 到通道 15 分配给当前用户组，用户可根据实际使用对通道进行配置，owner 表示设置当前用户组，赋值为 ug2 表示用户组 2 使用 ARM4 核。

```
used_for_rv {
 channels = <>;
 owner = <&ug1>;
};
```

可选更改配置项:used_for_rv 表示该配置在 RV 端生效，channels 表示通道分配，上述配置是未 dma 通道分配给当前用户组，用户可根据实际使用对通道进行配置，owner 表示设置当前用户组，赋值为 ug1 表示用户组 1 使用 E907 单核。

2.5.5.4 使用示例

每个 dma 的通道可灵活分配给 linux 或 rv, 只需给 channels 赋值对应通道即可，如下配置是将 DMA1 的通道 0 到通道 1 分配给了 linux 端，将 DMA1 的通道 2 到通道 7 分配给 rv 端。

```
 dma_resource_manager@1 {
  controller = <&dma1>;
  hw_isolation;
  hw_user_group_num = <2>;
  used_for_linux {
   channels = <0>, <1>;
```

```
    owner = <&ug2>;
  };

  used_for_rv {
    channels = <2>, <3>，<4>, <5>, <6>, <7>;
    owner = <&ug1>;
  };
};
```

说明

DMA 通道隔离功能不是所有版型都支持，若该版型支持 DMA 通道隔离功能，则会在 board.dts 增加上述配置，通过查看是 board.dts 否有上述配置，可确认该版型是否支持 DMA 通道隔离功能。

2.6 模式

2.6.1 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。

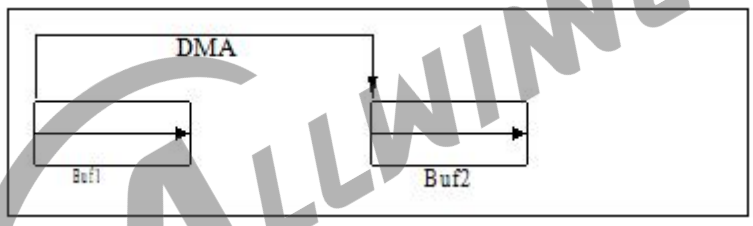

图 2-2: DMA Engine 内存拷贝示意图

2.6.2 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。

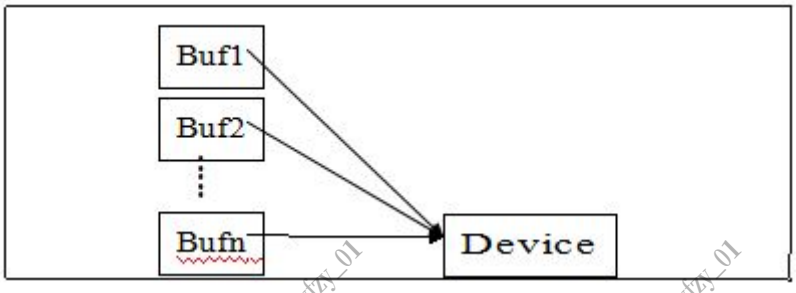

图 2-3: DMA Engine 散列拷贝示意图 (slave 与 master)

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。

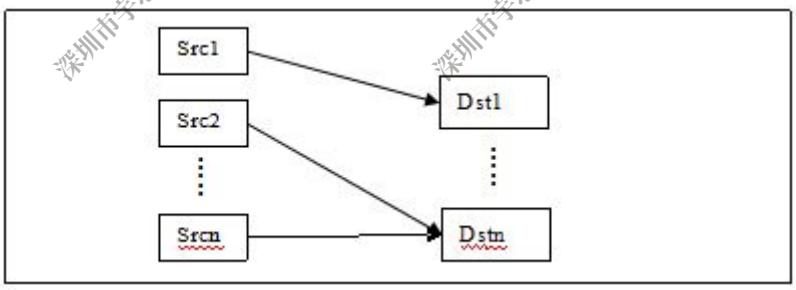

图 2-4: DMA Engine 散列拷贝示意图 (master 与 master)

2.6.3 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。

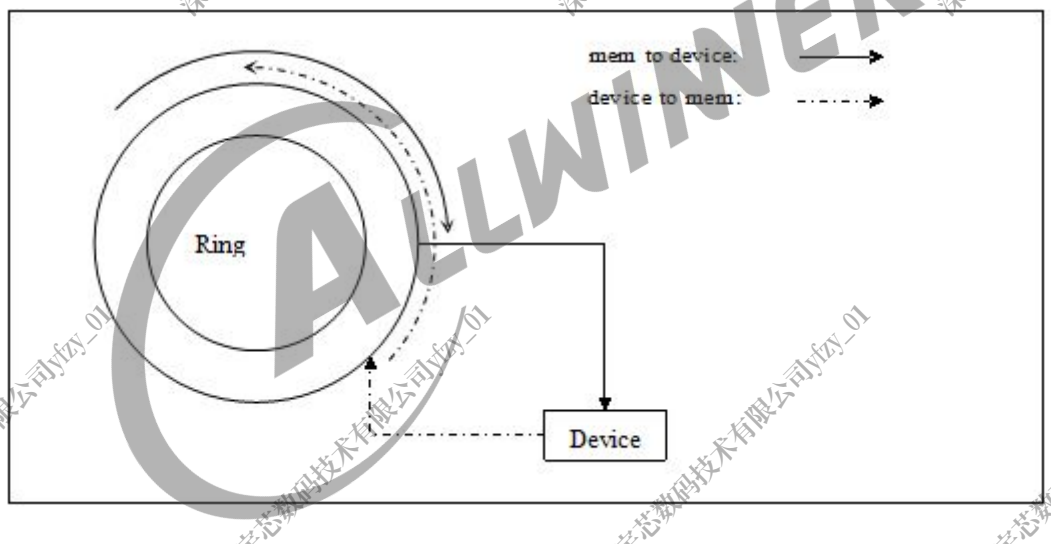

图 2-5: DMA Engine 循环拷贝示意图

注：NDMA 不支持循环缓存模式（cyclic）

2.7 模块接口说明

2.7.1 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。

2.7.2 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

2.7.3 dmaengine_slave_config

- 原型：int dmaengine_slave_config(struct dma_chan *chan, struct dma_slave_config *config)
- 作用：散列模式下，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。

:::note

dma_slave_config结构说明如下：

:::

```
struct dma_slave_config {
    enum dma_transfer_direction direction;
    dma_addr_t src_addr;
    dma_addr_t dst_addr;
    enum dma_slave_buswidth src_addr_width;
    enum dma_slave_buswidth dst_addr_width;
```

```
    u32 src_maxburst;
    u32 dst_maxburst;
    bool device_fc;
    unsigned int slave_id;
};

direction:   传输方向，取值MEM_TO_DEV DEV_TO_MEM

src_addr:   源地址，必须是物理地址

dst_addr:   目的地址，必须是物理地址

src_addr_width:   源数据宽度，对应byte整数倍，取值1，2，4，8;在代码中对应DMA_SLAVE_BUSWIDTH_1_BYTE~
    DMA_SLAVE_BUSWIDTH_8_BYTE

dst_addr_width:   目的数据宽度，取值同上

src_max_burst:   源突发长度，取值1，4，8

dst_max_burst:   目的突发长度，取值同上

slave_id: 从通道id号，此处用作DRQ的设置，具体取值参照dma datasheet中典型应用章节drq表使用。
```

2.7.4 dmaengine_prep_dma_cyclic

- 函数原型：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 direction, unsigned long flags)
- 作用：循环模式下，配置通道描述符信息
- 参数：
  - chan: 指向要操作的DMA通道句柄。
  - buf_addr: 本次传输 ring buff 地址。
  - buf_len：本次传输的 ring buff 长度
  - period_len：本次传输每包数据的长度
  - direction： 取值 MEM_TO_DEV DEV_TO_MEM
  - flags: 用于控制本次传输操作，具体详见 dmaengine.h 中 dma_ctrl_flags
- 返回：
  - 成功，返回结构体指针。
  - 失败，返回 NULL。

:::note

传输描述符介绍：

:::

```
struct dma_async_tx_descriptor {
    dma_cookie_t cookie;
    enum dma_ctrl_flags flags; /* not a 'long' to pack with cookie */
    dma_addr_t phys;
    struct dma_chan *chan;
    dma_cookie_t (*tx_submit)(struct dma_async_tx_descriptor *tx);
    dma_async_tx_callback callback;
    void *callback_param;
};
```

cookie：　本次传输的cookie，在此通道上唯一

tx_submit：　本次传输的提交执行函数

callback：　传输完成后的回调函数

callback_param：回调函数的参数

2.7.5 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。

2.7.6 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。

2.7.7 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。

:::warning 此功能会丢弃未开始的传输。:::

2.7.8 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。

2.7.9 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。

2.7.10 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。

3 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

3.1 基本流程

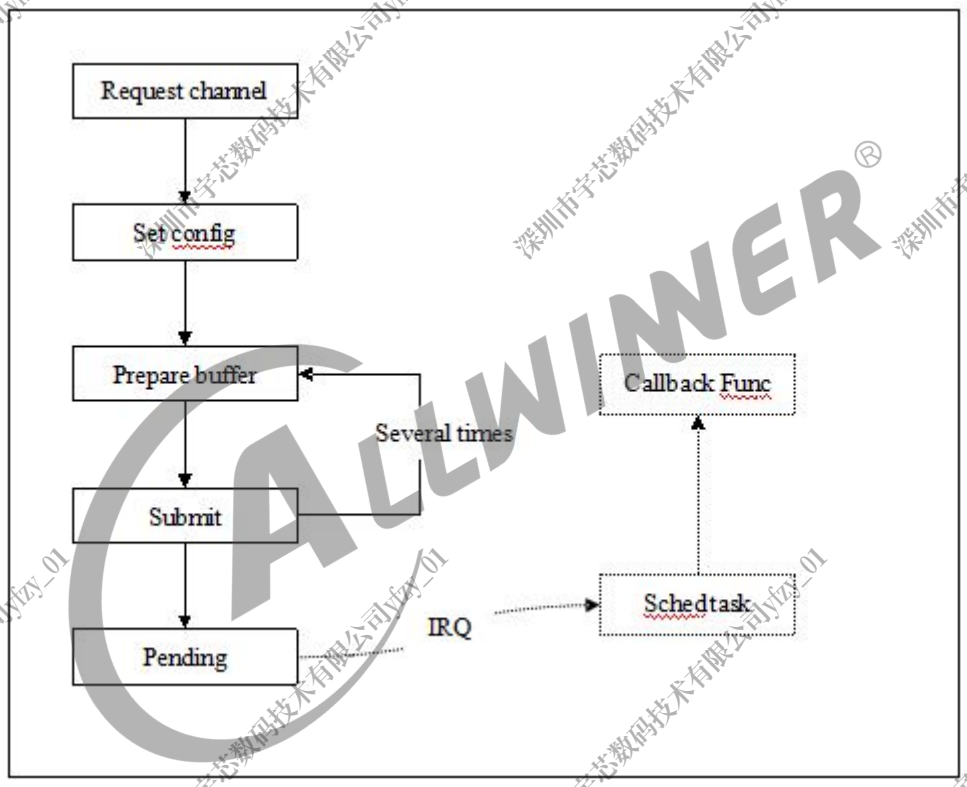

图 3-1: DMA Engine 使用流程

其中：Callback Func 在本包（循环模式）/笔（散列模式）数据输出完成时，在 dma 中断中会调用使用 dma 模块注册的 callback 函数，此时模块需根据业务需求在 callback 函数中处理。

3.2 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

4 使用范例

```
    struct dma_chan *chan;
    dma_cap_mask_t mask;
    dma_cookie_t cookie;
    struct dma_slave_config config;
    struct dma_tx_state state;
    struct dma_async_tx_descriptor *tx = NULL;
    void *src_buf;
    dma_addr_t src_dma;

dma_cap_zero(mask);
dma_cap_set(DMA_SLAVE, mask);
dma_cap_set(DMA_CYCLIC, mask);

/* 申请一个可用通道 */
chan = dma_request_channel(dt->mask, NULL, NULL);
if (!chan){
   return -EINVAL;
}

    src_buf = kmalloc(1024*4, GFP_KERNEL);
    if (!src_buf) {
       dma_release_channel(chan);
       return -EINVAL;
    }

    /* 映射地址用DMA访问 */
    src_dma = dma_map_single(NULL, src_buf, 1024*4, DMA_TO_DEVICE);

config.direction = DMA_MEM_TO_DEV;
config.src_addr = src_dma;
config.dst_addr = 0x01c;
config.src_addr_width = DMA_SLAVE_BUSWIDTH_2_BYTES;
config.dst_addr_width = DMA_SLAVE_BUSWIDTH_2_BYTES;
config.src_maxburst = 1;
config.dst_maxburst = 1;
config.slave_id = sunxi_slave_id(DRQDST_AUDIO_CODEC, DRQSRC_SDRAM);

dmaengine_slave_config(chan, &config);

    tx = dmaengine_pre_dma_cyclic(chan, scr_dma, 1024*4, 1024, DMA_MEM_TO_DEV,
                DMA_PREP_INTERRUPT | DMA_CTRL_ACK);

    /* 设置回调函数 */
    tx->callback = dma_callback;
    tx->callback = NULL;

    /* 提交及启动传输 */
    cookie = dmaengine_submit(tx);
    dma_async_issue_pending(chan);
```

5 常见问题调试方法

5.1 利用 sunxi_dump 读写相应寄存器

```
cd /sys/class/sunxi_dump/
1.查看一个寄存器
 echo 0x03002000 > dump ;cat dump

结果如下：
cupid-p1:/sys/class/sunxi_dump # echo 0x03002000 > dump ;cat dump
0x00000022

2.写值到寄存器上
 echo 0x03002000 0x1 > write ;cat write

3.查看一片连续寄存器
 echo 0x03002000,0x03002fff > dump;cat dump

结果如下：
cupid-p1:/sys/class/sunxi_dump # echo 0x03002000,0x03002fff > dump;cat dump

0x0000000003002000: 0x00000022 0x00000000 0x00000000 0x00000000
0x0000000003002010: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002020: 0x000000ff 0x00000000 0x00000007 0x00000000
0x0000000003002030: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002040: 0x00030000 0x00000000 0x00000000 0x00000000
0x0000000003002050: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002060: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002070: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002080: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002090: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030020a0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030020b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030020c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030020d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030020e0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030020f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002100: 0x00000000 0x00000000 0xfc0000e0 0x83460240
0x0000000003002110: 0xfc106500 0x05096020 0x00000b80 0x00010008
0x0000000003002120: 0x00000000 0x00000000 0x0000000c 0xfc0000c0
0x0000000003002130: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002140: 0x00000000 0x00000000 0xfc0001e0 0x83430240
0x0000000003002150: 0xfc506200 0x05097030 0x00000e80 0x00010008
0x0000000003002160: 0x00000000 0x00000000 0x0000000c 0xfc0001c0
0x0000000003002170: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002180: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002190: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030021a0: 0x00000000 0x00000001 0x00000000 0x00000000
0x00000000030021b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030021c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030021d0: 0x00000000 0x00000000 0x00000000 0x00000000
```

```
0x00000000030021e0: 0x00000000 0x00000001 0x00000000 0x00000000
0x00000000030021f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002200: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002210: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002220: 0x00000000 0x00000001 0x00000000 0x00000000
0x0000000003002230: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002240: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002250: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002260: 0x00000000 0x00000001 0x00000000 0x00000000
0x0000000003002270: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002280: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002290: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030022a0: 0x00000000 0x00000001 0x00000000 0x00000000
0x00000000030022b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030022c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030022d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030022e0: 0x00000000 0x00000001 0x00000000 0x00000000
0x00000000030022f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002300: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002310: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002320: 0x00000000 0x00000001 0x00000000 0x00000000
0x0000000003002330: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002340: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002350: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002360: 0x00000000 0x00000001 0x00000000 0x00000000
0x0000000003002370: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002380: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003002390: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030023a0: 0x00000000 0x00000001 0x00000000 0x00000000
0x00000000030023b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030023c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030023d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030023e0: 0x00000000 0x00000001 0x00000000 0x00000000
```

通过上述步骤，可以查看每个 DMA 通道寄存器的数据，即对照寄存器手册，检查通道是否使能、源地址、目的地址、搬运剩余数据量等等，例：

(1) 通过偏移地址 0x100，即上述的 0x03002100 寄存器，可以确认当前通道的使能位为 0；

(2) 通过偏移地址 0x110，即上述的 0x03002110 寄存器，可以确认当前 DMA 通道搬运数据的源地址为 0xfc106500；

通过上述方法，可以清楚当前 DMA 工作的状态是否符合预期，或者查看问题所在。

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





（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。