# Linux驱动\_DMA

Linux下DMA也有框架dmaengine，遗憾的是2440上并没有适配这个框架。但是也需先了解下dma功能的大致使用。

2410的DMA支持四类DMA传输：系统总线到系统总线(ASB/AHB to ASB/AHB)，系统总线到外设总线(ASB/AHB to APB)，外设总线到系统总线(APB to ASB/AHB),外设总线到外设总线(APB to APB)。

2410共有四条DMA通道，每条通道5个请求源。  
Ch0:nXDREQ0,UART0,SDI,Timer,USB EP1  
Ch1: nXDREQ1,UART1,I2SSDI,SPI0,USB EP2  
Ch2:I2SSDO,I2SSDI,SDI,Timer, USB EP3  
Ch3:UART1,SDI,SPI1,Timer, USB EP4

## 1. DMA寄存器

DCON寄存器定义了DMA的一些属性，下面来分析下。

(1)Req&Ack：DMA请求和应答的协议有两种，Demard mode 和 Handshake mode。  
两者对Request和Ack的时序定义有所不同：在Demard模式下，如果DMA完成一次请求如果Request仍然有效，那么DMA就认为这是下一次DMA请求；在Handshake模式下，DMA完成一次请求后等待Request信号无效，然后把ACK也置无效，再等待下一次Request。这个设计外部DMA请求时可能要用到。具体对应DCON[31]。

(2)Atomic transfer：指的是DMA的单次原子操作，它可以是Unit模式（传输1个data size），也可以是burst模式（传输4个data size），具体对应DCON[28]。

(3)Data Size：指的是单次原子操作的数据位宽，8、16、32，具体对应DCON[21:20]。

(4)Request Source：DMA请求的来源有两种，软件＆硬件模块，由DCON[23]控制；当为前者时，由软件对DMASKTRIG寄存器的位0置位触发一次DMA 操作。当为后者时，具体来源由DCON[26:24]控制，不同硬件模块的某时间触发一次DMA操作，具体要见不同的硬件模块。

(5)DMA service mode：DMA的工作模式有两种，单一服务模式＆整体服务模式。前一模式下，一次DMA请求完成一项原子操作，并且transfer count的值减1。后一模式下，一次DMA请求完成一批原子操作，直到transfer count等于0表示完成一次整体服务。具体对应DCON[27]。  
在单一服务模式下，不使用传统的DMA计数器，三个DMA状态被顺序执行一次后停止，等待DMA 请求再一次来临后再重新开始另一次循环。   
在整体服务模式下，使用传统的DMA 计数器，状态机会停留在状态三，直到DMA计数器的值减为零，再回到状态一，等待下一次DMA请求。  
Whole service模式下，这种读、写操作重复进行直到计数器（CURR\_TC）变为0；而Single service模式下读和写操作只进行一次。

(6)RELOAD：在reload模式下，当transfer count的值变为零时，将自动加src、dst、TC的值加载到CURR\_DST、CURR\_SRC、CURR\_TC，并开始一次新的DMA传输。该模式一般和整体服务模式一起使用，也就是说当一次整体服务开始后，src、dst、TC的值都已经被加载，因此可以更改为下一次。

(7)传输总长度：DMA一次整体服务传输的总长度为：  
    Data Size × Atomic transfer size × TC（字节）。

## 2. 使用DMA在内存间传输数据

开辟两块内存，在内存之间传递数据。

先开辟两块连续的物理内存。

src **=** dma\_alloc\_writecombine**(NULL,** BUF\_SIZE**,** **&**src\_phys**,** GFP\_KERNEL**);**

dst **=** dma\_alloc\_writecombine**(NULL,** BUF\_SIZE**,** **&**dst\_phys**,** GFP\_KERNEL**);**

下面开始设置dma寄存器。

设置源物理地址，物理内存位于AHB总线，这里要设置地址递增，也就是一次传输完成后，下次传输的地址会递增。

dma\_regs**->**disrc **=** src\_phys**;** /\* 源的物理地址 \*/

dma\_regs**->**disrcc **=** **(**0**<<**1**)** **|** **(**0**<<**0**);** /\* 源位于AHB总线, 源地址递增 \*/

设置目的物理地址，bit2设置为0表示计数减少到0时，会触发dma中断。

dma\_regs**->**didst **=** dst\_phys**;** /\* 目的的物理地址 \*/

dma\_regs**->**didstc **=** **(**0**<<**2**)** **|** **(**0**<<**1**)** **|** **(**0**<<**0**);** /\* 目的位于AHB总线, 目的地址递增 \*/

下面设置DMA的传输方式。

dma\_regs**->**dcon **=** **(**1**<<**30**)|(**1**<<**29**)|(**0**<<**28**)|(**1**<<**27**)|(**0**<<**23**)|(**0**<<**20**)|(**BUF\_SIZE**<<**0**);** /\* 使能中断,单个传输,软件触发, \*/

bit31：由于是内存之间的传输，采用默认的demand模式。

bit30：由于内存SDRAM都在AHP总线上， 所以需要设置为1。

bit29：当CURR\_TC变成0的时候，需要产生中断。所以设置为1。

bit28：一次传输一个unit。

bit27：whole service mode，一次DMA请求完成一批原子操作，直到CURR\_TC变成0。

bit26~24：由于采用软件触发方式，可以不设置。

bit23：采用软件触发dma的方式。

bit22：reload是开启的。

bit21~20：一个unit为一个byte。

bit19~0：需要传输的长度。

设置好寄存器，接下来就要手动启动DMA传输。

dma\_regs**->**dmasktrig **=** **(**1**<<**1**)** **|** **(**1**<<**0**);**

bit1：dma channel turned on。当reload为off时，CURR\_TC变成0时，该dma channel会自动关闭。

bit0：手动启动一次dma传输。

当启动一次DMA传输，在内存之间就会传递BUF\_SIZE长度的数据，传递完成产生dma中断。

## 3. 使用DMA在内存和IISFIFO之间传递数据。

当播放或者录音时，使用dma在内存和IISFIFO之间传递数据就非常的有必要了。我们需要分析下2410自带的DMA操作函数。

首先要使用IISFIFO的DMA操作，需要先使能IIS相关的寄存器。

iisfcon **|=** S3C2410\_IISFCON\_RXDMA**;** //Set RX FIFO acces mode to DMA

iisfcon **|=** S3C2410\_IISFCON\_TXDMA**;** //Set RX FIFO acces mode to DMA

iiscon **|=** S3C2410\_IISCON\_TXDMAEN**;** //Enable TX DMA service request

iiscon **|=** S3C2410\_IISCON\_RXDMAEN**;** //Enable RX DMA service request

### 3.1 DMA驱动注册

这里使用了系统设备的概念，通过内核中源码的注释我们看看什么是系统设备。系统设备与驱动模型有一点不同，他们不需要动态的驱动绑定，也不能被探测，并且不属于任何类型的外围总线。对系统设备我们仍然有驱动的概念，因为我们仍想执行在这些设备上执行基本的操作。

注册系统设备类s3c2440\_sysclass：

struct sysdev\_class s3c2440\_sysclass **=** **{**

set\_kset\_name**(**"s3c2440-core"**),**

**.**suspend **=** s3c244x\_suspend**,**

**.**resume **=** s3c244x\_resume

**};**

static int \_\_init s3c2440\_core\_init**(**void**)**

**{**

**return** sysdev\_class\_register**(&**s3c2440\_sysclass**);**

**}**

将dma驱动注册到设备类下：

static struct sysdev\_driver s3c2440\_dma\_driver **=** **{**

**.**add **=** s3c2440\_dma\_add**,**

**};**

static int \_\_init s3c2440\_dma\_init**(**void**)**

**{**

**return** sysdev\_driver\_register**(&**s3c2440\_sysclass**,** **&**s3c2440\_dma\_driver**);**

**}**

注册系统设备s3c2440\_sysdev：

static struct sys\_device s3c2440\_sysdev **=** **{**

**.**cls **=** **&**s3c2440\_sysclass**,**

**};**

sysdev\_register**(&**s3c2440\_sysdev**);**

注意系统设备这个结构体，里边封装了一个系统设备类，系统设备是一个虚拟设备。

在注册系统设备时，会调用dma注册的add函数。

int sysdev\_register**(**struct sys\_device **\***sysdev**){**

    。。。。。。。。。。。。

    /\* Notify class auxillary drivers \*/

    list\_for\_each\_entry**(**drv**,** **&**cls**->**drivers**,** entry**)** **{**

        /\*为这个设备调用了设备类下所有驱动的add函数\*/

        if **(**drv**->**add**)**

            drv**->**add**(**sysdev**);**

**}**

    。。。。。。。。。。。。。

**}**

dma注册的add函数为s3c2440\_dma\_add。

static int s3c2410\_dma\_add**(**struct sys\_device **\***sysdev**)**

**{**

s3c2410\_dma\_init**();**

s3c24xx\_dma\_order\_set**(&**s3c2410\_dma\_order**);**

**return** s3c24xx\_dma\_init\_map**(&**s3c2410\_dma\_sel**);**

**}**

在s3c2440\_dma\_add中初始化了4个dma通道，并注册了虚拟通道和物理通道的对应关系。

#### 3.1.1 s3c2410\_dma\_init

初始化4个dma通道，每个通道用s3c2410\_dma\_chan结构体描述，该函数初始化了每个通道对应的中断号和寄存器地址。

int s3c2410\_dma\_init**(**void**)**

**{**

 /\*4个通道，中断号为IRQ\_DMA0,每一个通道的寄存器覆盖的地址范围为0x40\*/

**return** s3c24xx\_dma\_init**(**4**,** IRQ\_DMA0**,** 0x40**);**

**}**

int \_\_init s3c24xx\_dma\_init**(**unsigned int channels**,** unsigned int irq**,**

                unsigned int stride**)**

**{**

    /\*每一个通道用一个s3c2410\_dma\_chan结构体描述\*/

    struct s3c2410\_dma\_chan **\***cp**;**

    int channel**;**

    int ret**;**

    /\*dma\_channels是一个全局变量，用来存放通道数量\*/

    dma\_channels **=** channels**;**

    /\*获得DMA寄存器的虚拟起始地址\*/

    dma\_base **=** ioremap**(**S3C24XX\_PA\_DMA**,** stride **\*** channels**);**

    /\*分配一个高速缓冲区，以后用来分配s3c2410\_dma\_buf\*/

    dma\_kmem **=** kmem\_cache\_create**(**"dma\_desc"**,**

                     sizeof**(**struct s3c2410\_dma\_buf**),** 0**,**

                     SLAB\_HWCACHE\_ALIGN**,**

                     s3c2410\_dma\_cache\_ctor**);**

    for **(**channel **=** 0**;** channel **<** channels**;**  channel**++)** **{**

        cp **=** **&**s3c2410\_chans**[**channel**];**

        memset**(**cp**,** 0**,** sizeof**(**struct s3c2410\_dma\_chan**));**

        /\*对通道的结构体进行初始化\*/

        cp**->**number **=** channel**;**            //通道号

        cp**->**irq    **=** channel **+** irq**;**      //通道中断号

        cp**->**regs   **=** dma\_base **+** **(**channel **\*** stride**);**  //通道寄存器基址

        /\* point current stats somewhere \*/

        cp**->**stats  **=** **&**cp**->**stats\_store**;**

        cp**->**stats\_store**.**timeout\_shortest **=** LONG\_MAX**;**

        /\*设置加载的超时时间\*/

        cp**->**load\_timeout **=** 1**<<**18**;**

        printk**(**"DMA channel %d at %p, irq %d/n"**,**

               cp**->**number**,** cp**->**regs**,** cp**->**irq**);**

**}**

    return 0**;**

**}**

#### 3.1.2 s3c24xx\_dma\_order\_set

该对象主要是用于保存DMA通道的可用关系的,IISDI可用通道为1通道和2通道。  
开发人员想要发起一次IIS\_IN的DMA时，它给出一个虚拟通道号DMACH\_I2S\_IN，而后内核再根据s3c2440\_dma\_order中的信息去选择物理通道1或者2。

static struct s3c24xx\_dma\_order \_\_initdata s3c2440\_dma\_order **=** **{**

**.**channels **=** **{**

**[**DMACH\_SDI**]** **=** **{**

**.**list **=** **{**

**[**0**]** **=** 3 **|** DMA\_CH\_VALID**,**

**[**1**]** **=** 2 **|** DMA\_CH\_VALID**,**

**[**2**]** **=** 1 **|** DMA\_CH\_VALID**,**

**[**3**]** **=** 0 **|** DMA\_CH\_VALID**,**

**},**

**},**

**[**DMACH\_I2S\_IN**]** **=** **{**

**.**list **=** **{**

**[**0**]** **=** 1 **|** DMA\_CH\_VALID**,**

**[**1**]** **=** 2 **|** DMA\_CH\_VALID**,**

**},**

**},**

**[**DMACH\_I2S\_OUT**]** **=** **{**

**.**list **=** **{**

**[**0**]** **=** 2 **|** DMA\_CH\_VALID**,**

**[**1**]** **=** 1 **|** DMA\_CH\_VALID**,**

**},**

**},**

**[**DMACH\_PCM\_IN**]** **=** **{**

**.**list **=** **{**

**[**0**]** **=** 2 **|** DMA\_CH\_VALID**,**

**[**1**]** **=** 1 **|** DMA\_CH\_VALID**,**

**},**

**},**

**[**DMACH\_PCM\_OUT**]** **=** **{**

**.**list **=** **{**

**[**0**]** **=** 1 **|** DMA\_CH\_VALID**,**

**[**1**]** **=** 3 **|** DMA\_CH\_VALID**,**

**},**

**},**

**[**DMACH\_MIC\_IN**]** **=** **{**

**.**list **=** **{**

**[**0**]** **=** 3 **|** DMA\_CH\_VALID**,**

**[**1**]** **=** 2 **|** DMA\_CH\_VALID**,**

**},**

**},**

**},**

**};**

int \_\_init s3c24xx\_dma\_order\_set**(**struct s3c24xx\_dma\_order **\***ord**)**

**{**

struct s3c24xx\_dma\_order **\***nord **=** dma\_order**;**

**if** **(**nord **==** **NULL)**

nord **=** kmalloc**(sizeof(**struct s3c24xx\_dma\_order**),** GFP\_KERNEL**);**

**if** **(**nord **==** **NULL)** **{**

printk**(**KERN\_ERR "no memory to store dma channel order\n"**);**

**return** **-**ENOMEM**;**

**}**

dma\_order **=** nord**;**

memcpy**(**nord**,** ord**,** **sizeof(**struct s3c24xx\_dma\_order**));**

**return** 0**;**

**}**

#### 3.1.3 s3c24xx\_dma\_init\_map

为了统一管理，内核中没有使用直接的DMA通道号，也就是说内核中，使用通道0，未必指的是S3C2440手册中给出的DMA-CH0，内核中的通道号使用的是虚拟通道号，是由实际（物理）通道号映射得来的，这个映射关系就保存在struct s3c24xx\_dma\_map和s3c24xx\_dma\_order结构中。两者都可以通过虚拟通道号映射得到物理通道号，但优先使用s3c24xx\_dma\_order的映射。因为s3c24xx\_dma\_order是用于特定板子的映射。而struct s3c24xx\_dma\_map用于芯片的映射。

static struct s3c24xx\_dma\_map \_\_initdata s3c2440\_dma\_mappings**[]** **=** **{**

**[**DMACH\_XD0**]** **=** **{**

**.**name **=** "xdreq0"**,**

**.**channels**[**0**]** **=** S3C2410\_DCON\_CH0\_XDREQ0 **|** DMA\_CH\_VALID**,**

**},**

**[**DMACH\_XD1**]** **=** **{**

**.**name **=** "xdreq1"**,**

**.**channels**[**1**]** **=** S3C2410\_DCON\_CH1\_XDREQ1 **|** DMA\_CH\_VALID**,**

**},**

**[**DMACH\_SDI**]** **=** **{**

**.**name **=** "sdi"**,**

**.**channels**[**0**]** **=** S3C2410\_DCON\_CH0\_SDI **|** DMA\_CH\_VALID**,**

**.**channels**[**1**]** **=** S3C2440\_DCON\_CH1\_SDI **|** DMA\_CH\_VALID**,**

**.**channels**[**2**]** **=** S3C2410\_DCON\_CH2\_SDI **|** DMA\_CH\_VALID**,**

**.**channels**[**3**]** **=** S3C2410\_DCON\_CH3\_SDI **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_IIS **+** S3C2410\_IISFIFO**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_IIS **+** S3C2410\_IISFIFO**,**

**},**

**[**DMACH\_SPI0**]** **=** **{**

**.**name **=** "spi0"**,**

**.**channels**[**1**]** **=** S3C2410\_DCON\_CH1\_SPI **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_SPI **+** S3C2410\_SPTDAT**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_SPI **+** S3C2410\_SPRDAT**,**

**},**

**[**DMACH\_SPI1**]** **=** **{**

**.**name **=** "spi1"**,**

**.**channels**[**3**]** **=** S3C2410\_DCON\_CH3\_SPI **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_SPI **+** 0x20 **+** S3C2410\_SPTDAT**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_SPI **+** 0x20 **+** S3C2410\_SPRDAT**,**

**},**

**[**DMACH\_UART0**]** **=** **{**

**.**name **=** "uart0"**,**

**.**channels**[**0**]** **=** S3C2410\_DCON\_CH0\_UART0 **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_UART0 **+** S3C2410\_UTXH**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_UART0 **+** S3C2410\_URXH**,**

**},**

**[**DMACH\_UART1**]** **=** **{**

**.**name **=** "uart1"**,**

**.**channels**[**1**]** **=** S3C2410\_DCON\_CH1\_UART1 **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_UART1 **+** S3C2410\_UTXH**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_UART1 **+** S3C2410\_URXH**,**

**},**

**[**DMACH\_UART2**]** **=** **{**

**.**name **=** "uart2"**,**

**.**channels**[**3**]** **=** S3C2410\_DCON\_CH3\_UART2 **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_UART2 **+** S3C2410\_UTXH**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_UART2 **+** S3C2410\_URXH**,**

**},**

**[**DMACH\_TIMER**]** **=** **{**

**.**name **=** "timer"**,**

**.**channels**[**0**]** **=** S3C2410\_DCON\_CH0\_TIMER **|** DMA\_CH\_VALID**,**

**.**channels**[**2**]** **=** S3C2410\_DCON\_CH2\_TIMER **|** DMA\_CH\_VALID**,**

**.**channels**[**3**]** **=** S3C2410\_DCON\_CH3\_TIMER **|** DMA\_CH\_VALID**,**

**},**

**[**DMACH\_I2S\_IN**]** **=** **{**

**.**name **=** "i2s-sdi"**,**

**.**channels**[**1**]** **=** S3C2410\_DCON\_CH1\_I2SSDI **|** DMA\_CH\_VALID**,**

**.**channels**[**2**]** **=** S3C2410\_DCON\_CH2\_I2SSDI **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**from **=** S3C2410\_PA\_IIS **+** S3C2410\_IISFIFO**,**

**},**

**[**DMACH\_I2S\_OUT**]** **=** **{**

**.**name **=** "i2s-sdo"**,**

**.**channels**[**0**]** **=** S3C2440\_DCON\_CH0\_I2SSDO **|** DMA\_CH\_VALID**,**

**.**channels**[**2**]** **=** S3C2410\_DCON\_CH2\_I2SSDO **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2410\_PA\_IIS **+** S3C2410\_IISFIFO**,**

**},**

**[**DMACH\_PCM\_IN**]** **=** **{**

**.**name **=** "pcm-in"**,**

**.**channels**[**0**]** **=** S3C2440\_DCON\_CH0\_PCMIN **|** DMA\_CH\_VALID**,**

**.**channels**[**2**]** **=** S3C2440\_DCON\_CH2\_PCMIN **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**from **=** S3C2440\_PA\_AC97 **+** S3C\_AC97\_PCM\_DATA**,**

**},**

**[**DMACH\_PCM\_OUT**]** **=** **{**

**.**name **=** "pcm-out"**,**

**.**channels**[**1**]** **=** S3C2440\_DCON\_CH1\_PCMOUT **|** DMA\_CH\_VALID**,**

**.**channels**[**3**]** **=** S3C2440\_DCON\_CH3\_PCMOUT **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**to **=** S3C2440\_PA\_AC97 **+** S3C\_AC97\_PCM\_DATA**,**

**},**

**[**DMACH\_MIC\_IN**]** **=** **{**

**.**name **=** "mic-in"**,**

**.**channels**[**2**]** **=** S3C2440\_DCON\_CH2\_MICIN **|** DMA\_CH\_VALID**,**

**.**channels**[**3**]** **=** S3C2440\_DCON\_CH3\_MICIN **|** DMA\_CH\_VALID**,**

**.**hw\_addr**.**from **=** S3C2440\_PA\_AC97 **+** S3C\_AC97\_MIC\_DATA**,**

**},**

**[**DMACH\_USB\_EP1**]** **=** **{**

**.**name **=** "usb-ep1"**,**

**.**channels**[**0**]** **=** S3C2410\_DCON\_CH0\_USBEP1 **|** DMA\_CH\_VALID**,**

**},**

**[**DMACH\_USB\_EP2**]** **=** **{**

**.**name **=** "usb-ep2"**,**

**.**channels**[**1**]** **=** S3C2410\_DCON\_CH1\_USBEP2 **|** DMA\_CH\_VALID**,**

**},**

**[**DMACH\_USB\_EP3**]** **=** **{**

**.**name **=** "usb-ep3"**,**

**.**channels**[**2**]** **=** S3C2410\_DCON\_CH2\_USBEP3 **|** DMA\_CH\_VALID**,**

**},**

**[**DMACH\_USB\_EP4**]** **=** **{**

**.**name **=** "usb-ep4"**,**

**.**channels**[**3**]** **=** S3C2410\_DCON\_CH3\_USBEP4 **|** DMA\_CH\_VALID**,**

**},**

**};**

static void s3c2440\_dma\_select**(**struct s3c2410\_dma\_chan **\***chan**,**

struct s3c24xx\_dma\_map **\***map**)**

**{**

**//dcon默认值**

chan**->**dcon **=** map**->**channels**[**chan**->**number**]** **&** **~**DMA\_CH\_VALID**;**

**}**

static struct s3c24xx\_dma\_selection \_\_initdata s3c2440\_dma\_sel **=** **{**

**.**select **=** s3c2440\_dma\_select**,**

**.**dcon\_mask **=** 7 **<<** 24**,**

**.**map **=** s3c2440\_dma\_mappings**,**

**.**map\_size **=** ARRAY\_SIZE**(**s3c2440\_dma\_mappings**),**

**};**

S3C24xx的DMA基本[架构](http://lib.csdn.net/base/architecture)已经建立起来的。至此，驱动开发的程序员就可以使用S3C24XX DMA子系统所提供的API进行开发的。

### 3.2 播放DMA操作

#### 3.2.1 s3c2410\_dma\_request

s3c2410\_dma\_request**(**s**->**dma\_ch**,** **&**s3c2410iis\_dma\_out**,** **NULL);**

static struct s3c2410\_dma\_client s3c2410iis\_dma\_out**=** **{**

**.**name **=** "I2SSDO"**,**

**};**

s**->**dma\_ch **=** DMACH\_I2S\_OUT

s3c2410\_dma\_request源码如下：

int s3c2410\_dma\_request**(**enum dma\_ch channel**,**struct s3c2410\_dma\_client **\***client**,**void **\***dev**)**

**{**

struct s3c2410\_dma\_chan **\***chan**;**

unsigned long flags**;**

int err**;**

local\_irq\_save**(**flags**);**

/\*通过虚拟DMA通道映射得到实际通道描述信息\*/

chan **=** s3c2410\_dma\_map\_channel**(**channel**);**

dbg\_showchan**(**chan**);**

/\*标记正在使用\*/

chan**->**client **=** client**;**

chan**->**in\_use **=** 1**;**

/\*如果中断尚未注册就注册中断\*/

**if** **(!**chan**->**irq\_claimed**)** **{**

/\*标记中断已经注册\*/

chan**->**irq\_claimed **=** 1**;**

local\_irq\_restore**(**flags**);**

/\*注册中断\*/

err **=** request\_irq**(**chan**->**irq**,** s3c2410\_dma\_irq**,** IRQF\_DISABLED**,**

client**->**name**,** **(**void **\*)**chan**);**

local\_irq\_save**(**flags**);**

local\_irq\_restore**(**flags**);**

**return** chan**->**number **|** DMACH\_LOW\_LEVEL**;**

**}**

该函数所做的工作有   
(1)通过虚拟通道号映射得到物理通道号，并且通过物理通道号索引得到通道描述结构struct s3c2410\_dma\_chan \*chan 。  
(2)根据struct s3c2410\_dma\_chan \*chan的描述，进行中断注册，并且标记为通道正在使用状态。

通过虚拟通道号映射得到物理通道号的工作在s3c2410\_dma\_map\_channel中完成，该函数只是遍历了s3c2410\_dma\_order和s3c2440\_dma\_mappings找到映射的物理通道号，而后再通过物理通道号在s3c2410\_chans中索引得到通道描述结构。并且调用了(dma\_sel.select)(dmach, ch\_map)回调函数。

#### 3.2.2 s3c2410\_dma\_devconfig

函数s3c2410\_dma\_devconfig所做的工作是配置DMA传输方向，和DMA设备地址。而**源数据和源地址在发起DMA请求的时候才会给出**。在该函数中，首先将配置的信息保存到对应的s3c2410\_dma\_chan结构中，而后将配置写入寄存器中以配置DMA。

#define BUF\_ON\_APB (ON\_APB | ADDR\_FIX)

channel **=** DMACH\_I2S\_OUT**;**

source **=** S3C2410\_DMASRC\_MEM**;**

hwcfg **=** BUF\_ON\_APB**;**

devaddr **=** 0x55000010**;**

s3c2410\_dma\_devconfig**(**channel**,** source**,** hwcfg**,** devaddr**);**

int s3c2410\_dma\_devconfig**(**int channel**,**enum s3c2410\_dmasrc source**,**int hwcfg**,**unsigned long devaddr**)**

**{**

struct s3c2410\_dma\_chan **\***chan **=** lookup\_dma\_channel**(**channel**);**

chan**->**source **=** source**;**

chan**->**dev\_addr **=** devaddr**;**

**switch** **(**source**)** **{**

**case** S3C2410\_DMASRC\_HW**:**

/\* source is hardware \*/

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DISRCC**,** hwcfg **&** 3**);**

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DISRC**,** devaddr**);**

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DIDSTC**,** **(**0**<<**1**)** **|** **(**0**<<**0**));**

chan**->**addr\_reg **=** dma\_regaddr**(**chan**,** S3C2410\_DMA\_DIDST**);**

**return** 0**;**

**case** S3C2410\_DMASRC\_MEM**:**

/\* source is memory \*/

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DISRCC**,** **(**0**<<**1**)** **|** **(**0**<<**0**));**

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DIDST**,** devaddr**);**

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DIDSTC**,** hwcfg **&** 3**);**

chan**->**addr\_reg **=** dma\_regaddr**(**chan**,** S3C2410\_DMA\_DISRC**);**

**return** 0**;**

**}**

**return** **-**EINVAL**;**

**}**

#### 3.2.3 s3c2410\_dma\_config

函数s3c2410\_dma\_config用于配置传输单元大小，实际上是根据用户提供的参数在配置DCON寄存器，但这里注意的是，在这里只是将配置信息写入对应的s3c2410\_dma\_chan结构，而没有真正写入硬件寄存器当中。dcon配置寄存器用于配置DMA具体行为，是一个十分重要的配置环节。dcon寄存器将在数据被装载到DMA通道时写入硬件寄存器。

dcon **=** S3C2410\_DCON\_HANDSHAKE**|**S3C2410\_DCON\_SYNC\_PCLK**|**S3C2410\_DCON\_INTREQ**|**S3C2410\_DCON\_TSZUNIT**|**S3C2410\_DCON\_SSERVE**|**S3C2410\_DCON\_CH2\_I2SSDO**|**S3C2410\_DCON\_HWTRIG**;** // VAL: 0xa0800000;

s3c2410\_dma\_config**(**channel**,** 2**,** dcon**);**

下面分析一下dcon的设置。

采用Handshake方式，选择PCLK作为DREQ/DACK同步，开启中断，每次传输一个unit，single service mode，触发源I2SSDO，硬件触发方式，每个unit 2byte。

为什么选择单一服务模式，因为这里的dma传输依赖I2SSDO触发，每触发一次传输一个字节，而不是传送完所有的字节。

#### 3.2.4 s3c2410\_dma\_set\_buffdone\_fn

s3c2410\_dma\_cbfn\_t callback\_fn**;** /\* buffer done callback \*/

s3c2410\_dma\_opfn\_t op\_fn**;** /\* channel op callback \*/

其中成员callback\_fn在装载数据是被调用。而op\_fn在DMA操作完成时被调用。其调用过程在后续分析中会讲到。

s3c2410\_dma\_set\_buffdone\_fn**(**channel**,** audio\_dmaout\_done\_callback**);**

static void audio\_dmaout\_done\_callback**(**struct s3c2410\_dma\_chan **\***ch**,** void **\***buf**,** int size**,**enum s3c2410\_dma\_buffresult result**)**

**{**

audio\_buf\_t **\***b **=** **(**audio\_buf\_t **\*)** buf**;**

up**(&**b**->**sem**);**

wake\_up**(&**b**->**sem**.**wait**);**

**}**

在这里回调函数的作用是取消传输队列的锁定，可以继续使用而不会阻塞。

#### 3.2.5 s3c2410\_dma\_setflags

flags **=** S3C2410\_DMAF\_AUTOSTART**;**

s3c2410\_dma\_setflags**(**channel**,** flags**);**

当有数据需要传输时，自动开始dma传输。

#### 3.2.6 写入操作

当前面的设置完成后，就可以开始进行dma的传输了。

**发送队列：**

在s3c2410\_dma\_chan结构中有三个成员用于维护该通道的发送队列：

struct s3c2410\_dma\_buf **\***curr

struct s3c2410\_dma\_buf **\***next

struct s3c2410\_dma\_buf **\***end

三个成员都指向了struct s3c2410\_dma\_buf结构，该结构记录了需要传送的数据起始地址，以及长度信息。其中curr成员指向的是当前需要发送缓冲，next则指向下一个发送的缓冲，而end指向最后一个发送缓冲。当每一个缓冲区数据被发送完成之后就会触发DMA中断，在中断中更新curr和next，取得下一个发送的缓冲。如此反复知道整个队列中的缓冲都被发送完毕。

**DMA状态和装载状态：**

s3c2410\_dma\_chan结构中有两个成员:

enum s3c2410\_dma\_loadst load\_state

enum s3c2410\_dma\_state state

定义如下：

enum s3c2410\_dma\_state **{**

S3C2410\_DMA\_IDLE**,**

S3C2410\_DMA\_RUNNING**,**

S3C2410\_DMA\_PAUSED

**};**

enum s3c2410\_dma\_loadst **{**

S3C2410\_DMALOAD\_NONE**,**

S3C2410\_DMALOAD\_1LOADED**,**

S3C2410\_DMALOAD\_1RUNNING**,**

S3C2410\_DMALOAD\_1LOADED\_1RUNNING**,**

**};**

其中s3c2410\_dma\_state枚举用来表示DMA控制器的状态，与硬件上的DMA控制器状态相对应，分别有运行，空闲，暂停三种状态。

而s3c2410\_dma\_loadst用于表示DMA缓冲装载状态。前文说过，要使用DMA传输数据，就要先将欲传输的DMA数据缓冲挂入DMA通道的缓冲队列中，DMA控制器在中断中依次取出加载到硬件然后进行发送。

**+** S3C2410\_DMA\_NONE表示当前通道没有任何需要发送的缓冲。

**+** S3C2410\_DMA\_1LOADED表示当前已经有一个缓冲挂入该通道，但硬件还没开始发送缓冲。

**+** S3C2410\_DMA\_1RUNNING表示当前有缓冲正在被硬件传输，传输正在进行时。

**+** S3C2410\_DMA\_1LOADED\_1RUNNING则表示当前有一个缓冲正在传输，并且还有一个缓冲正在等待被传输

这几种状态之间的转化关系我们可以在s3c2410\_dma\_irq和s3c2410\_dma\_loadbuffer函数中直观地看到。

顺藤摸瓜，从write函数开始分析。

##### 3.2.6.1 audio\_setup\_buf

在开始传输之前，先使用dma\_alloc\_coherent分配好DMA传输所需的连续物理空间。这里的物理空间分为了8段，每段8192个字节。每一段内存空间用audio\_buf\_t结构体描述，这8个缓冲区循环使用，每个缓冲区有信号量来做互斥操作。

#define AUDIO\_NBFRAGS\_DEFAULT 8

#define AUDIO\_FRAGSIZE\_DEFAULT 8192

static int audio\_setup\_buf**(**audio\_stream\_t **\*** s**)**

**{**

int frag**;**

int dmasize **=** 0**;**

char **\***dmabuf **=** 0**;**

dma\_addr\_t dmaphys **=** 0**;**

s**->**nbfrags **=** audio\_nbfrags**;** //片段的数量8

s**->**fragsize **=** audio\_fragsize**;**  //片段的大小8192

s**->**buffers **=** **(**audio\_buf\_t **\*)**kmalloc**(sizeof(**audio\_buf\_t**)** **\*** s**->**nbfrags**,** GFP\_KERNEL**);**

memset**(**s**->**buffers**,** 0**,** **sizeof(**audio\_buf\_t**)** **\*** s**->**nbfrags**);**

**for** **(**frag **=** 0**;** frag **<** s**->**nbfrags**;** frag**++)** **{**

audio\_buf\_t **\***b **=** **&**s**->**buffers**[**frag**];**

//如果没有分配dma传输空间，则分配

**if** **(!**dmasize**)** **{**

//总长度为8192\*8

dmasize **=** **(**s**->**nbfrags **-** frag**)** **\*** s**->**fragsize**;**

**do** **{**

dmabuf **=** dma\_alloc\_coherent**(NULL,** dmasize**,** **&**dmaphys**,** GFP\_KERNEL**|**GFP\_DMA**);**

**if** **(!**dmabuf**)**

dmasize **-=** s**->**fragsize**;**

**}** **while** **(!**dmabuf **&&** dmasize**);**

**if** **(!**dmabuf**)**

**goto** err**;**

b**->**master **=** dmasize**;**

**}**

//每段内存空间的虚拟起始地址

b**->**start **=** dmabuf**;**

//每段内存空间的物理起始地址

b**->**dma\_addr **=** dmaphys**;**

//初始化信号量

sema\_init**(&**b**->**sem**,** 1**);**

//更新下一段内存空间的起始地址

dmabuf **+=** s**->**fragsize**;**

dmaphys **+=** s**->**fragsize**;**

dmasize **-=** s**->**fragsize**;**

**}**

//更新当前使用的空间id和对应的audio\_buf\_t结构体

s**->**buf\_idx **=** 0**;**

s**->**buf **=** **&**s**->**buffers**[**0**];**

**return** 0**;**

**}**

##### 3.2.6.2 smdk2410\_audio\_write

当上层传递过来音频数据，每收集到8192个字节，就开始发起dma传输。

static ssize\_t smdk2410\_audio\_write**(**struct file **\***file**,** const char **\***buffer**,**size\_t count**,** loff\_t **\*** ppos**)**

**{**

const char **\***buffer0 **=** buffer**;**

audio\_stream\_t **\***s **=** **&**output\_stream**;**

//如果没有分配dma物理空间，则分配dma传输使用的物理空间，分8段，每段8192个字节。

**if** **(!**s**->**buffers **&&** audio\_setup\_buf**(**s**))**

**return** **-**ENOMEM**;**

//需要4字节对齐

count **&=** **~**0x03**;**

**while** **(**count **>** 0**)** **{**

//从分配的缓冲区中，取出当前可用的一个dma物理空间段

audio\_buf\_t **\***b **=** s**->**buf**;**

//如果非阻塞打开，对指定的物理空间段尝试上锁，失败退出

//如果是阻塞打开，则指定的物理空间段会上锁，当8个物理空间段都写满时，则需要等待。

**if** **(**file**->**f\_flags **&** O\_NONBLOCK**)** **{**

ret **=** **-**EAGAIN**;**

**if** **(**down\_trylock**(&**b**->**sem**))**

**break;**

**}** **else** **{**

ret **=** **-**ERESTARTSYS**;**

**if** **(**down\_interruptible**(&**b**->**sem**))**

**break;**

**}**

**if** **(**audio\_channels **==** 2**)** **{**

//计算指定的dma物理空间段剩余的空间

chunksize **=** s**->**fragsize **-** b**->**size**;**

//如果此次写入的数据该物理空间还能存下

**if** **(**chunksize **>** count**)**

chunksize **=** count**;**

//拷贝数据到缓冲区

copy\_from\_user**(**b**->**start **+** b**->**size**,** buffer**,** chunksize**);**

//更新缓冲区当前数据长度

b**->**size **+=** chunksize**;**

**}**

//更新buffer位置和剩余写入数据

buffer **+=** chunksize**;**

count **-=** chunksize**;**

//如果当前缓冲区size还不足8192，进入到下一个循环，直到凑足8192个字节。

**if** **(**b**->**size **<** s**->**fragsize**)** **{**

up**(&**b**->**sem**);**

**break;**

**}**

//如果凑足了8192个字节，则发起dma传输。

**if((**ret **=** s3c2410\_dma\_enqueue**(**s**->**dma\_ch**,** **(**void **\*)** b**,** b**->**dma\_addr**,** b**->**size**)))** **{**

printk**(**PFX"dma enqueue failed.\n"**);**

**return** ret**;**

**}**

//将该段物理空间size清0,并使用下一个物理空间段。

//因为该物理空间段正在使用，下一次写入将使用下一个物理空间。

b**->**size **=** 0**;**

NEXT\_BUF**(**s**,** buf**);**

**}**

**}**

##### 3.2.6.3 s3c2410\_dma\_enqueue

s3c2410\_dma\_enqueue**(**s**->**dma\_ch**,** **(**void **\*)** b**,** b**->**dma\_addr**,** b**->**size**)**

b为audio\_buf\_t结构体

b**->**dma\_addr为dma传输要使用的物理地址

b**->**size为dma传输长度

int s3c2410\_dma\_enqueue**(**unsigned int channel**,** void **\***id**,**dma\_addr\_t data**,** int size**)**

**{**

struct s3c2410\_dma\_chan **\***chan **=** lookup\_dma\_channel**(**channel**);**

struct s3c2410\_dma\_buf **\***buf**;**

unsigned long flags**;**

//分配发送队列

buf **=** kmem\_cache\_alloc**(**dma\_kmem**,** GFP\_ATOMIC**);**

buf**->**next **=** **NULL;**

buf**->**data **=** buf**->**ptr **=** data**;** //buf->data dma传输物理地址起始 buf->ptr dma传输到哪儿了

buf**->**size **=** size**;** //传输长度

buf**->**id **=** id**;** //id为audio\_buf\_t结构体

buf**->**magic **=** BUF\_MAGIC**;** //定义一个魔数

local\_irq\_save**(**flags**);**

**if** **(**chan**->**curr **==** **NULL)** **{**

//如果当前没有传输队列，更新curr end next传输队列状态

chan**->**curr **=** buf**;**

chan**->**end **=** buf**;**

chan**->**next **=** **NULL;**

**}** **else** **{**

//如果当前有传输队列，则更新队列信息

chan**->**end**->**next **=** buf**;**

chan**->**end **=** buf**;**

**}**

//如果有必要，更新next也指向buf

**if** **(**chan**->**next **==** **NULL)**

chan**->**next **=** buf**;**

**if** **(**chan**->**state **==** S3C2410\_DMA\_RUNNING**)** **{**

//如果此时dma传输打开

//只有在DMA处于运行状态时才能装载数据

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_1LOADED **&&** 1**)** **{**

//S3C2410\_DMA\_1LOADED表示当前已经有一个缓冲挂入该通道，但硬件还没开始发送缓冲。

//等待dma引擎装载成功

**if** **(**s3c2410\_dma\_waitforload**(**chan**,** \_\_LINE\_\_**)** **==** 0**)** **{**

//检查装载是否超时

local\_irq\_restore**(**flags**);**

**return** **-**EINVAL**;**

**}**

**}**

//只有当前没有DMA传输，或者DMA传输正在进行时，才可以装载下一个数据队列

//装载数据，也就是将传输缓冲中的数据地址和长度等等写入寄存器

**while** **(**s3c2410\_dma\_canload**(**chan**)** **&&** chan**->**next **!=** **NULL)** **{**

s3c2410\_dma\_loadbuffer**(**chan**,** chan**->**next**);**

**}**

**}** **else** **if** **(**chan**->**state **==** S3C2410\_DMA\_IDLE**)** **{**

**if** **(**chan**->**flags **&** S3C2410\_DMAF\_AUTOSTART**)** **{**

//启动dma传输，并装载一次数据 s3c2410\_dma\_start

s3c2410\_dma\_ctrl**(**chan**->**number **|** DMACH\_LOW\_LEVEL**,** S3C2410\_DMAOP\_START**);**

**}**

**}**

local\_irq\_restore**(**flags**);**

**return** 0**;**

**}**

S3C24XX DMA框架将框架用户需要传输的DMA数据通过链表的形式组织起来，由各个通道的struct s3c2410\_dma\_cha实例进行维护。启动传输之后，在每次中断中取出链表头指向的缓冲继续发送，链表头也由此往下滑动，如此反复直到所有缓冲都被发送完毕。

DMA的装载策略：当一次DMA传输已经开始了，此时则可以装载下一个传输队列，可以提前将DMA寄存器写好，等待当前DMA传输完成，如果reload开启了，则自动装载源目的地址和长度，并开始下一次传输。

如果有一个队列正在做dma传输，第二个队列也加载好了，那么第三个队列要装载，必须等待dma传输完成，并且第二个队列装载成功。

##### 3.2.6.4 s3c2410\_dma\_start

static int s3c2410\_dma\_start**(**struct s3c2410\_dma\_chan **\***chan**)**

**{**

unsigned long tmp**;**

unsigned long flags**;**

local\_irq\_save**(**flags**);**

//设置dma状态为运行

chan**->**state **=** S3C2410\_DMA\_RUNNING**;**

//如果当前通道没有任何dma传输,检查是否有队列需要传输

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_NONE**)** **{**

**if** **(**chan**->**next **==** **NULL)** **{**

printk**(**KERN\_ERR "dma%d: channel has nothing loaded\n"**,**

chan**->**number**);**

chan**->**state **=** S3C2410\_DMA\_IDLE**;**

local\_irq\_restore**(**flags**);**

**return** **-**EINVAL**;**

**}**

//装载数据到dma传输通道

s3c2410\_dma\_loadbuffer**(**chan**,** chan**->**next**);**

**}**

dbg\_showchan**(**chan**);**

//打开中断

**if** **(!**chan**->**irq\_enabled**)** **{**

enable\_irq**(**chan**->**irq**);**

chan**->**irq\_enabled **=** 1**;**

**}**

//开始传输

tmp **=** dma\_rdreg**(**chan**,** S3C2410\_DMA\_DMASKTRIG**);**

tmp **&=** **~**S3C2410\_DMASKTRIG\_STOP**;**

tmp **|=** S3C2410\_DMASKTRIG\_ON**;**

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DMASKTRIG**,** tmp**);**

s3c2410\_dma\_call\_op**(**chan**,** S3C2410\_DMAOP\_START**);**

//假如还有传输队列等待传输

**if** **(**chan**->**next **!=** **NULL)** **{**

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_1LOADED**)** **{**

//假如当前通道装载了数据但是未开始，需要等待dma引擎装载成功

**if** **(**s3c2410\_dma\_waitforload**(**chan**,** \_\_LINE\_\_**)** **==** 0**)** **{**

pr\_debug**(**"%s: buff not yet loaded, no more todo\n"**,**

\_\_FUNCTION\_\_**);**

**}** **else** **{**

//数据装载成功，开始dma传输，状态更新为S3C2410\_DMALOAD\_1RUNNING

chan**->**load\_state **=** S3C2410\_DMALOAD\_1RUNNING**;**

//装载下一个传输队列

s3c2410\_dma\_loadbuffer**(**chan**,** chan**->**next**);**

**}**

**}** **else** **if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_1RUNNING**)** **{**

//直接装载下一个传输队列

s3c2410\_dma\_loadbuffer**(**chan**,** chan**->**next**);**

**}**

**}**

local\_irq\_restore**(**flags**);**

**return** 0**;**

**}**

##### 3.2.6.5 s3c2410\_dma\_loadbuffer

static inline int s3c2410\_dma\_loadbuffer**(**struct s3c2410\_dma\_chan **\***chan**,**struct s3c2410\_dma\_buf **\***buf**)**

**{**

unsigned long reload**;**

//有数据挂入dma通道，还未开始传输

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_1LOADED**)** **{**

dmawarn**(**"load\_state is S3C2410\_DMALOAD\_1LOADED\n"**);**

**}**

//一个传输正在进行，还有一个传输正在等待

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_1LOADED\_1RUNNING**)** **{**

dmawarn**(**"state is S3C2410\_DMALOAD\_1LOADED\_1RUNNING\n"**);**

**}**

//假如当前通道没有任何需要发送的缓冲

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_NONE**)** **{**

//如果没有下一个需要传输的队列，则不需要reload

reload **=** **(**buf**->**next **==** **NULL)** **?** S3C2410\_DCON\_NORELOAD **:** 0**;**

**}** **else** **{**

//如果有数据正在传输，打开reload

reload **=** S3C2410\_DCON\_AUTORELOAD**;**

**}**

**if** **((**buf**->**data **&** 0xf0000000**)** **!=** 0x30000000**)** **{**

dmawarn**(**"dmaload: buffer is %p\n"**,** **(**void **\*)**buf**->**data**);**

**}**

//将要传输的物理地址写入到dma对应寄存器

writel**(**buf**->**data**,** chan**->**addr\_reg**);**

//写入dcon，传输开始

dma\_wrreg**(**chan**,** S3C2410\_DMA\_DCON**,**

chan**->**dcon **|** reload **|** **(**buf**->**size**/**chan**->**xfer\_unit**));**

//更新传输队列为下一个

chan**->**next **=** buf**->**next**;**

//更新状态

**switch** **(**chan**->**load\_state**)** **{**

**case** S3C2410\_DMALOAD\_NONE**:**

//如果当前通道没有传输队列，则更新为有传输队列正在进行

chan**->**load\_state **=** S3C2410\_DMALOAD\_1LOADED**;**

**break;**

**case** S3C2410\_DMALOAD\_1RUNNING**:**

//如果当前有传输队列，则更新为一个正在传输，还有一个传输队列在等待

chan**->**load\_state **=** S3C2410\_DMALOAD\_1LOADED\_1RUNNING**;**

**break;**

**}**

**return** 0**;**

**}**

##### 3.2.6.6 s3c2410\_dma\_irq

static irqreturn\_t s3c2410\_dma\_irq**(**int irq**,** void **\***devpw**)**

**{**

struct s3c2410\_dma\_chan **\***chan **=** **(**struct s3c2410\_dma\_chan **\*)**devpw**;**

struct s3c2410\_dma\_buf **\***buf**;**

buf **=** chan**->**curr**;**

dbg\_showchan**(**chan**);**

**switch** **(**chan**->**load\_state**)** **{**

//如果dma传输正在进行，并没有下一个队列，更新为dma传输完毕

**case** S3C2410\_DMALOAD\_1RUNNING**:**

chan**->**load\_state **=** S3C2410\_DMALOAD\_NONE**;**

**break;**

//如果队列装载好了，但是并未开始，则更新为dma传输完毕

**case** S3C2410\_DMALOAD\_1LOADED**:**

chan**->**load\_state **=** S3C2410\_DMALOAD\_NONE**;**

**break;**

//如果一个队列正在传输，还有一个队列在等待，则更新为一个队列已经装载了，但是未开始

**case** S3C2410\_DMALOAD\_1LOADED\_1RUNNING**:**

chan**->**load\_state **=** S3C2410\_DMALOAD\_1LOADED**;**

**break;**

**case** S3C2410\_DMALOAD\_NONE**:**

printk**(**KERN\_ERR "dma%d: IRQ with no loaded buffer?\n"**,**

chan**->**number**);**

**break;**

**}**

**if** **(**buf **!=** **NULL)** **{**

//更新curr队列指向

chan**->**curr **=** buf**->**next**;**

buf**->**next **=** **NULL;**

//检查魔数是否匹配

**if** **(**buf**->**magic **!=** BUF\_MAGIC**)** **{**

printk**(**KERN\_ERR "dma%d: %s: buf %p incorrect magic\n"**,**

chan**->**number**,** \_\_FUNCTION\_\_**,** buf**);**

**return** IRQ\_HANDLED**;**

**}**

//当dma传输完毕，释放当前队列的占用

s3c2410\_dma\_buffdone**(**chan**,** buf**,** S3C2410\_RES\_OK**);**

//释放dma\_kmem

s3c2410\_dma\_freebuf**(**buf**);**

**}** **else** **{**

**}**

//假如还有传输队列，并且dma还在运行

**if** **(**chan**->**next **!=** **NULL** **&&** chan**->**state **!=** S3C2410\_DMA\_IDLE**)** **{**

unsigned long flags**;**

**switch** **(**chan**->**load\_state**)** **{**

**case** S3C2410\_DMALOAD\_1RUNNING**:**

/\* don't need to do anything for this state \*/

**break;**

**case** S3C2410\_DMALOAD\_NONE**:**

/\* can load buffer immediately \*/

**break;**

**case** S3C2410\_DMALOAD\_1LOADED**:**

//等待当前装载队列开始dma传输

**if** **(**s3c2410\_dma\_waitforload**(**chan**,** \_\_LINE\_\_**)** **==** 0**)** **{**

/\* flag error? \*/

printk**(**KERN\_ERR "dma%d: timeout waiting for load (%s)\n"**,**

chan**->**number**,** \_\_FUNCTION\_\_**);**

**return** IRQ\_HANDLED**;**

**}**

**break;**

**case** S3C2410\_DMALOAD\_1LOADED\_1RUNNING**:**

**goto** no\_load**;**

**default:**

printk**(**KERN\_ERR "dma%d: unknown load\_state in irq, %d\n"**,**

chan**->**number**,** chan**->**load\_state**);**

**return** IRQ\_HANDLED**;**

**}**

local\_irq\_save**(**flags**);**

//再装载一个队列

s3c2410\_dma\_loadbuffer**(**chan**,** chan**->**next**);**

local\_irq\_restore**(**flags**);**

**}** **else** **{**

//如果没有数据需要传输了，关闭reload标志，传输完成便会关闭dma通道。

s3c2410\_dma\_lastxfer**(**chan**);**

//如果没有队列需要load了，则直接关闭dma传输通道。

**if** **(**chan**->**load\_state **==** S3C2410\_DMALOAD\_NONE**)** **{**

s3c2410\_dma\_ctrl**(**chan**->**number **|** DMACH\_LOW\_LEVEL**,**

S3C2410\_DMAOP\_STOP**);**

**}**

**}**

no\_load**:**

**return** IRQ\_HANDLED**;**

**}**

## 4.注意

在后面的内核版本中，sysdev\_class\_register 函数已经被替换成了subsys\_interface\_register。