

努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处



<http://blog.csdn.net/woshixingaaa/archive/2011/06/02/6462089.aspx>

首先介绍一下 DMA，S3C2440A 支持位于系统总线和外围总线之间的 4 通道 DMA 控制器，每个通道都可以在系统总线或外围总线上的设备之间传输数据。每个通道可以对下面 4 种情况进行传输：

- 1.源和目的都在系统总线上
- 2.源在系统总线而目的在外围总线
- 3.源在外围总线而目的在系统总线
- 4.源和目的都在外围总线

下图是请求源为硬件模式时的每个通道的请求源：

Table 8-1. DMA Request Sources for Each Channel

	Source0	Source1	Source2	Source3	Source4	Source5	Source6
Ch-0	nXDREQ0	UART0	SDI	Timer	USB device EP1	I2SSDO	PCMIN
Ch-1	nXDREQ1	UART1	I2SSDI	SPI0	USB device EP2	PCMOUT	SDI
Ch-2	I2SSDO	I2SSDI	SDI	 Timer	USB device EP3	PCMIN	MICIN
Ch-3	UART2	SDI	SPI1	 Timer	USB device EP4	MICIN	PCMOUT

DMA 使用 3 个状态的有限状态机：

- 1.初始状态，DMA 等待 DMA 请求，一旦请求到达 DMA 进入状态 2，DMA ACK 与 INT REQ 为 0。
- 2.在这个状态，DMA ACK 置为 1 并且计数器 CURR_TC 的值被从 DCON[19:0]载入，注意 DMA ACK 保持为 1 直到它被清除。
- 3.在这个状态，处理 DMA 原子操作的主状态机被初始化。子状态机从源地址读取数据，然后写入目的地址。在这个操作中，要考虑数据的大小和传输的大小(单个/突发)，这个操作重复执行直到计数器(CURR_TC)变成 0 在全服务模式，而只执行一次在单服务模式。当子状态机结束每一次原子操作的时候主状态机减少 CURR_TC 的值。另外，主状态机发出 INT REQ 信号当 CURR_TC 变成 0 并且 DCON[29]位被置位 1 时。并且，清除 DMA ACK，如果下面两个条件之一满足的话：

- 1)在全服务模式下 CURR_TC 变成 0
- 2)在单服务模式下结束原子操作

注意在单服务模式下，主有限状态机的 3 个状态被执行然后停止，等待另一个



DMA REQ。如果 DMA REQ 到来，所有的 3 个状态被重复执行。所以在每次原子操作中 DMA ACK 被置位，然后又被清除。与之对比，在全服务模式，主有限状态机等在状态 3 直到 CURR_TC 变成 0。所以，DMA ACK 在传输期间被置位，到 TC 为 0 时被清除。

然而，不管哪个服务模式，INT REQ 被发出只有当 CURR_TC 变成 0 时。

如下图，是基本的 DMA 时序：

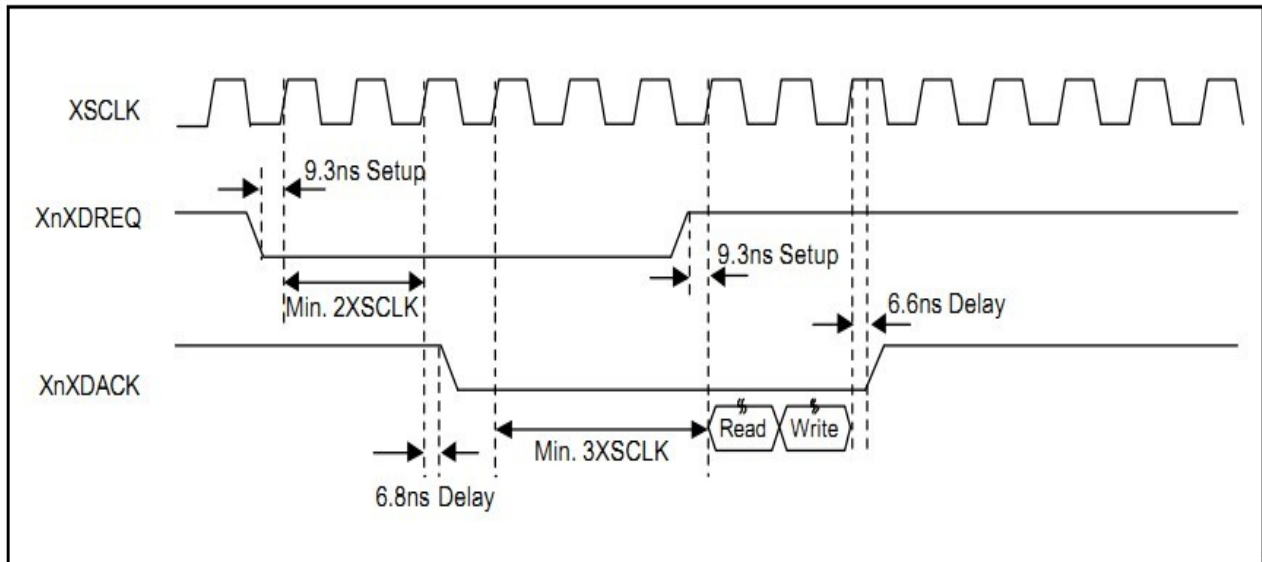


Figure 8-1. Basic DMA Timing Diagram

nXDREQ 生效后等待至少 2 个时钟周期,nXDACK 响应并开始生效，但要知道延时至少 3 个时钟周期，DMA 控制器才可获得总线的控制权，进行读写操作一次。

下面来分析内核 DMA 驱动源码：

首先来看一下 DMA 驱动是怎样注册的：

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。

在 arch/arm/plat-s3c24xx/s3c244x.c 中，注册了系统设备的类：

```
1. struct sysdev_class s3c2440_sysclass = {
2.     .name      = "s3c2440-core",
3.     .suspend   = s3c244x_suspend,
4.     .resume    = s3c244x_resume
5. };
6. static int __init s3c2440_core_init(void)
7. {
8.     return sysdev_class_register(&s3c2440_sysclass);
9. }
```

在 arch/arm/mach-s3c2410/dma.c 中，注册了 dma 的驱动：

```
1. #if defined(CONFIG_CPU_S3C2410)
2. static struct sysdev_driver s3c2410_dma_driver = {
3.     .add = s3c2410_dma_add,
4. };
```

把 dma 驱动注册到设备类下：

```
1. static int __init s3c2410_dma_drivinit(void)
2. {
3.     return sysdev_driver_register(&s3c2410_sysclass, &s3c2410_dma_driver);
4. }
```

先来看一下系统设备类：

```
1. struct sysdev_class {
2.     const char *name;
3.     struct list_head drivers;
4.
5.     /* Default operations for these types of devices */
6.     int (*shutdown)(struct sys_device *);
7.     int (*suspend)(struct sys_device *, pm_message_t state);
8.     int (*resume)(struct sys_device *);
9.     struct kset kset;
10.};
```

这个结构体有一个 drivers 双向循环链表，注册到这个类的驱动都挂在这里。

下面分析一下 dma 驱动是怎样注册的：

```
1. int sysdev_driver_register(struct sysdev_class *cls, struct sysdev_driver *drv)
2. {
3.     . . . . .
4.     if (cls && kset_get(&cls->kset)) {
5.         /*这里把这个驱动添加到了设备类的驱动链表上*/
6.         list_add_tail(&drv->entry, &cls->drivers);
7.
8.         /*如果驱动定义了 add 方法，则为类下的每个设备调用驱动的 add 方法*/
9.         if (drv->add) {
10.            struct sys_device *dev;
11.            list_for_each_entry(dev, &cls->kset.list, kobj.entry)
12.                drv->add(dev);
13.        }
14.    } else {
15.        err = -EINVAL;
16.        WARN(1, KERN_ERR "%s: invalid device class/n", __func__);
17.    }
18.    mutex_unlock(&sysdev_drivers_lock);
```

```
19. return err;
20.}
```

在 arch/arm/mach-s3c2440/s3c2440.c 中，注册了一个系统设备 s3c2440_sysdev，

```
1. static struct sys_device s3c2440_sysdev = {
2.     .cls = &s3c2440_sysclass,
3. };
4. int __init s3c2440_init(void)
5. {
6.     . . . . .
7.     return sysdev_register(&s3c2440_sysdev);
8. }
```

注意系统设备这个结构体，里边封装了一个系统设备类。

```
1. struct sys_device {
2.     u32 id;
3.     struct sysdev_class *cls;
4.     struct kobject kobj;
5. };
```

下面来看一下系统设备的注册，系统设备是一个虚拟设备，这里的目的是为了调用 driver 的 add 函数。

```
1. int sysdev_register(struct sys_device *sysdev){
2.     . . . . .
3.     /* Notify class auxillary drivers */
4.     list_for_each_entry(drv, &cls->drivers, entry) {
5.         /*为这个设备调用了设备类下所有驱动的 add 函数*/
6.         if (drv->add)
7.             drv->add(sysdev);
8.     }
9.     . . . . .
10.}
```

下面来分析一下这个 add 函数。看上边的那个 dma 驱动的结构体，指明了 add 函数为 s3c2410_dma_add：

```
1. static int __init s3c2410_dma_add(struct sys_device *sysdev)
2. {
3.     s3c2410_dma_init(); (一)
4.     s3c24xx_dma_order_set(&s3c2410_dma_order); (二)
5.     return s3c24xx_dma_init_map(&s3c2410_dma_sel); (三)
6. }
```

分别对 s3c2410_dma_add 中的 3 个函数进行分析：

(一)

```

1. int __init s3c2410_dma_init(void)
2. {
3.     /*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*/
4.     return s3c24xx_dma_init(4, IRQ_DMA0, 0x40);
5. }

```

```

1. int __init s3c24xx_dma_init(unsigned int channels, unsigned int irq,
2.     unsigned int stride)
3. {
4.     /*每一个通道用一个 s3c2410_dma_chan 结构体描述*/
5.     struct s3c2410_dma_chan *cp;
6.     int channel;
7.     int ret;
8.
9.     printk("S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics/n");
10.    /*dma_channels 是一个全局变量, 用来存放通道数量*/
11.    dma_channels = channels;
12.    /*获得 DMA 寄存器的虚拟起始地址*/
13.    dma_base = ioremap(S3C24XX_PA_DMA, stride * channels);
14.    if (dma_base == NULL) {
15.        printk(KERN_ERR "dma failed to remap register block/n");
16.        return -ENOMEM;
17.    }
18.    /*分配一个高速缓冲区, 以后用来分配 s3c2410_dma_buf*/
19.    dma_kmem = kmem_cache_create("dma_desc",
20.        sizeof(struct s3c2410_dma_buf), 0,
21.        SLAB_HWCACHE_ALIGN,
22.        s3c2410_dma_cache_ctor);
23.
24.    if (dma_kmem == NULL) {
25.        printk(KERN_ERR "dma failed to make kmem cache/n");
26.        ret = -ENOMEM;
27.        goto err;
28.    }
29.
30.    for (channel = 0; channel < channels; channel++) {
31.        cp = &s3c2410_chans[channel];
32.        memset(cp, 0, sizeof(struct s3c2410_dma_chan));
33.        /*对通道的结构体进行初始化*/
34.        cp->number = channel;    //通道号
35.        cp->irq = channel + irq;    //通道中断号
36.        cp->regs = dma_base + (channel * stride);    //通道寄存器基址
37.
38.        /* point current stats somewhere */

```

```

39.     cp->stats = &cp->stats_store;
40.     cp->stats_store.timeout_shortest = LONG_MAX;
41.
42.     /* basic channel configuration */
43.     /*设置加载的超时时间*/
44.     cp->load_timeout = 1<<18;
45.     printk("DMA channel %d at %p, irq %d/n",
46.           cp->number, cp->regs, cp->irq);
47. }
48. return 0;
49.
50. err:
51.     kmem_cache_destroy(dma_kmem);
52.     iounmap(dma_base);
53.     dma_base = NULL;
54.     return ret;
55.}

```

这里使用到了一个 s3c2410_dma_chan 结构体，struct s3c2410_dma_chan 记录 dma 通道信息，内容如下：

```

1. 151 struct s3c2410_dma_chan {
2. 152     /* channel state flags and information */
3. 153     unsigned char    number;    //dma 通道号,
4. 154     unsigned char    in_use;    //当前通道是否已经使用
5. 155     unsigned char    irq_claimed; // 有无 dma 中断
6. 156     unsigned char    irq_enabled; //是否使能了 dma 中断
7. 157     unsigned char    xfer_unit; //传输块大小
8. 158
9. 159     /* channel state */
10.160
11.161     enum s3c2410_dma_state state;
12.162     enum s3c2410_dma_loadst load_state;
13.163     struct s3c2410_dma_client *client;
14.164
15.165     /* channel configuration */
16.166     enum s3c2410_dmasrc source;
17.167     enum dma_ch req_ch;
18.168     unsigned long dev_addr;
19.169     unsigned long load_timeout;
20.170     unsigned int flags;    /* channel flags */
21.171
22.172     struct s3c24xx_dma_map *map;    /* channel hw maps */
23.173
24.174     /* channel's hardware position and configuration */
25.175     void __iomem *regs;    /* channels registers */
26.176     void __iomem *addr_reg; /* data address register */
27.177     unsigned int irq;    中断号
28.178     unsigned long dcon;    /默认控制寄存器的值

```

```

29.179
30.180  /* driver handles */
31.181  s3c2410_dma_cbfn_t callback_fn; 传输完成回调函数
32.182  s3c2410_dma_opfn_t op_fn;      操作完成回调函数*/
33.183
34.184  /* stats gathering */
35.185  struct s3c2410_dma_stats *stats;
36.186  struct s3c2410_dma_stats stats_store;
37.187
38.188  /* buffer list and information */
39.189  struct s3c2410_dma_buf *curr;      /* current dma buffer */
40.190  struct s3c2410_dma_buf *next;      /* next buffer to load */
41.191  struct s3c2410_dma_buf *end;      /* end of queue */dma 缓冲区链表
42.192
43.193  /* system device */
44.194  struct sys_device dev;
45.195 };

```

(二)

先看下边一个结构体，s3c2410_dma_order。这个是建立目标板 dma 源与硬件的 dma 通道的关联。

```

1. static struct s3c24xx_dma_order __initdata s3c2410_dma_order = {
2.     .channels = {
3.         [DMACH_SDI] = {
4.             .list = {
5.                 [0] = 3 | DMA_CH_VALID,
6.                 [1] = 2 | DMA_CH_VALID,
7.                 [2] = 0 | DMA_CH_VALID,
8.             },
9.         },
10.        [DMACH_I2S_IN] = {
11.            .list = {
12.                [0] = 1 | DMA_CH_VALID,
13.                [1] = 2 | DMA_CH_VALID,
14.            },
15.        },
16.    },
17.};

```

分析这里 SDI 可以是使用通道 3,2,0，为什么从大到小排列，是因为某些 dma 请求只能使用 dma0，dma1 等较小的通道号，比如外部总线 dma 只能只用 dma0，为了避免 sdi 占用，这里就采用的这种排列。

```

1. [DMACH_SDI] = {
2.     .list = {

```

```

3.     [0] = 3 | DMA_CH_VALID,
4.     [1] = 2 | DMA_CH_VALID,
5.     [2] = 0 | DMA_CH_VALID,
6. },
7. },

```

注意这个结构体是用__initdata 修饰的，所以在初始化后会被释放掉。下边这个函数重新分配内存保存这个结构体就是这个原因。

```

1. int __init s3c24xx_dma_order_set(struct s3c24xx_dma_order *ord)
2. {
3.     struct s3c24xx_dma_order *nord = dma_order;
4.
5.     if (nord == NULL)
6.         nord = kmalloc(sizeof(struct s3c24xx_dma_order), GFP_KERNEL);
7.
8.     if (nord == NULL) {
9.         printk(KERN_ERR "no memory to store dma channel order/n");
10.        return -ENOMEM;
11.    }
12.
13.    dma_order = nord;
14.    memcpy(nord, ord, sizeof(struct s3c24xx_dma_order));
15.    return 0;
16.}

```

(三)

```

1. struct s3c24xx_dma_map {
2.     const char *name; //DMA 源的名
3.     struct s3c24xx_dma_addr hw_addr; //源的物理地址
4.
5.     unsigned long channels[S3C2410_DMA_CHANNELS]; //DMA 通道信息
6.     unsigned long channels_rx[S3C2410_DMA_CHANNELS];
7. };
8.
9. struct s3c24xx_dma_selection {
10.    struct s3c24xx_dma_map *map; //记录了 struct s3c24xx_dma_map 数组的首地址
11.    unsigned long map_size; //struct s3c24xx_dma_map 数组的成员个数
12.    unsigned long dcon_mask; //dma 控制器掩码
13.
14.    void (*select)(struct s3c2410_dma_chan *chan,
15.        struct s3c24xx_dma_map *map); //源选择函数
16.
17.    void (*direction)(struct s3c2410_dma_chan *chan, //dma 方向

```



```

18.     struct s3c24xx_dma_map *map,
19.     enum s3c2410_dmasrc dir);
20.};

```

建立芯片本身的 dma 源与硬件 dma 通道的视图。

```

1. int __init s3c24xx_dma_init_map(struct s3c24xx_dma_selection *sel)
2. {
3.     struct s3c24xx_dma_map *nmap;
4.     size_t map_sz = sizeof(*nmap) * sel->map_size;
5.     int ptr;
6.
7.     nmap = kmalloc(map_sz, GFP_KERNEL);
8.     if (nmap == NULL)
9.         return -ENOMEM;
10.
11.     memcpy(nmap, sel->map, map_sz);
12.     memcpy(&dma_sel, sel, sizeof(*sel));
13.
14.     dma_sel.map = nmap;
15.
16.     for (ptr = 0; ptr < sel->map_size; ptr++)
17.         s3c24xx_dma_check_entry(nmap+ptr, ptr);
18.
19.     return 0;
20.}
21. static struct s3c24xx_dma_selection __initdata s3c2410_dma_sel = {
22.     .select = s3c2410_dma_select, //通道选择函数
23.     .dcon_mask = 7 << 24, //屏蔽 DMA 控制寄存器中用于选择请求源的位
24.     .map = s3c2410_dma_mappings, //dma 源与硬件 dma 通道的视图
25.     .map_size = ARRAY_SIZE(s3c2410_dma_mappings), //虚拟通道的数目
26.};
27. static void s3c2410_dma_select(struct s3c2410_dma_chan *chan,
28.     struct s3c24xx_dma_map *map)
29. {
30.     chan->dcon = map->channels[chan->number] & ~DMA_CH_VALID; //选择通道,并设置成请求模式
31.}
32. static struct s3c24xx_dma_map __initdata s3c2410_dma_mappings[] = {
33.     [DMACH_XD0] = {
34.         .name = "xdreq0",
35.         .channels[0] = S3C2410_DCON_CH0_XDREQ0 | DMA_CH_VALID,
36.     },
37.     [DMACH_XD1] = {
38.         .name = "xdreq1",
39.         .channels[1] = S3C2410_DCON_CH1_XDREQ1 | DMA_CH_VALID,

```

```

40. },
41. [DMACH_SDI] = {
42.     .name      = "sdi",
43.     .channels[0] = S3C2410_DCON_CH0_SDI | DMA_CH_VALID,
44.     .channels[2] = S3C2410_DCON_CH2_SDI | DMA_CH_VALID,
45.     .channels[3] = S3C2410_DCON_CH3_SDI | DMA_CH_VALID,
46.     .hw_addr.to = S3C2410_PA_IIS + S3C2410_IISFIFO,
47.     .hw_addr.from = S3C2410_PA_IIS + S3C2410_IISFIFO,
48. },
49. [DMACH_SPI0] = {
50.     .name      = "spi0",
51.     .channels[1] = S3C2410_DCON_CH1_SPI | DMA_CH_VALID,
52.     .hw_addr.to = S3C2410_PA_SPI + S3C2410_SPTDAT,
53.     .hw_addr.from = S3C2410_PA_SPI + S3C2410_SPRDAT,
54. },
55. [DMACH_SPI1] = {
56.     .name      = "spi1",
57.     .channels[3] = S3C2410_DCON_CH3_SPI | DMA_CH_VALID,
58.     .hw_addr.to = S3C2410_PA_SPI + 0x20 + S3C2410_SPTDAT,
59.     .hw_addr.from = S3C2410_PA_SPI + 0x20 + S3C2410_SPRDAT,
60. },
61. [DMACH_UART0] = {
62.     .name      = "uart0",
63.     .channels[0] = S3C2410_DCON_CH0_UART0 | DMA_CH_VALID,
64.     .hw_addr.to = S3C2410_PA_UART0 + S3C2410_UTXH,
65.     .hw_addr.from = S3C2410_PA_UART0 + S3C2410_URXH,
66. },
67. [DMACH_UART1] = {
68.     .name      = "uart1",
69.     .channels[1] = S3C2410_DCON_CH1_UART1 | DMA_CH_VALID,
70.     .hw_addr.to = S3C2410_PA_UART1 + S3C2410_UTXH,
71.     .hw_addr.from = S3C2410_PA_UART1 + S3C2410_URXH,
72. },
73. [DMACH_UART2] = {
74.     .name      = "uart2",
75.     .channels[3] = S3C2410_DCON_CH3_UART2 | DMA_CH_VALID,
76.     .hw_addr.to = S3C2410_PA_UART2 + S3C2410_UTXH,
77.     .hw_addr.from = S3C2410_PA_UART2 + S3C2410_URXH,
78. },
79. [DMACH_TIMER] = {
80.     .name      = "timer",
81.     .channels[0] = S3C2410_DCON_CH0_TIMER | DMA_CH_VALID,
82.     .channels[2] = S3C2410_DCON_CH2_TIMER | DMA_CH_VALID,
83.     .channels[3] = S3C2410_DCON_CH3_TIMER | DMA_CH_VALID,
84. },

```

```
85. [DMACH_I2S_IN] = {
86.     .name      = "i2s-sdi",
87.     .channels[1] = S3C2410_DCON_CH1_I2SSDI | DMA_CH_VALID,
88.     .channels[2] = S3C2410_DCON_CH2_I2SSDI | DMA_CH_VALID,
89.     .hw_addr.from = S3C2410_PA_IIS + S3C2410_IISFIFO,
90. },
91. [DMACH_I2S_OUT] = {
92.     .name      = "i2s-sdo",
93.     .channels[2] = S3C2410_DCON_CH2_I2SSDO | DMA_CH_VALID,
94.     .hw_addr.to = S3C2410_PA_IIS + S3C2410_IISFIFO,
95. },
96. [DMACH_USB_EP1] = {
97.     .name      = "usb-ep1",
98.     .channels[0] = S3C2410_DCON_CH0_USBEP1 | DMA_CH_VALID,
99. },
100. [DMACH_USB_EP2] = {
101.     .name      = "usb-ep2",
102.     .channels[1] = S3C2410_DCON_CH1_USBEP2 | DMA_CH_VALID,
103. },
104. [DMACH_USB_EP3] = {
105.     .name      = "usb-ep3",
106.     .channels[2] = S3C2410_DCON_CH2_USBEP3 | DMA_CH_VALID,
107. },
108. [DMACH_USB_EP4] = {
109.     .name      = "usb-ep4",
110.     .channels[3] = S3C2410_DCON_CH3_USBEP4 | DMA_CH_VALID,
111. },
112.};
```