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/2011/06/02/6462149.aspx

DMA 通道的使用：申请通道，申请中断，设置寄存器，安装回调函数，设置标志，将数据放入队列，最后就是调用 static int s3c2410_dma_start(struct s3c2410_dma_chan *chan)来开始 DMA 的传输了。

首先看通道的申请：

```c
int s3c2410_dma_request(unsigned int channel,
            struct s3c2410_dma_client *client,
            void *dev)
{
    struct s3c2410_dma_chan *chan;
    unsigned long flags;
    int err;

    pr_debug("dma%d: s3c2410_request_dma: client=%s, dev=%p/n",
         channel, client->name, dev);

    local_irq_save(flags);             //关中断
    /*找到一个有效的物理通道*/
    chan = s3c2410_dma_map_channel(channel);
    if (chan == NULL) {
        local_irq_restore(flags);
        return -EBUSY;
    }

    dbg_showchan(chan);
    /*设置通道的名字*/
    chan->client = client;
    /*设置通道的使用标志*/
    chan->in_use = 1;

    if (!chan->irq_claimed) {      //该中断没有被注册
        pr_debug("dma%d: %s : requesting irq %d/n",
             channel, __func__, chan->irq);

        chan->irq_claimed = 1;   //标记该中断被注
        local_irq_restore(flags); //开中断

        err = request_irq(chan->irq, s3c2410_dma_irq, IRQF_DISABLED,  //注册中断处理程序
                  client->name, (void *)chan);

        local_irq_save(flags);

        if (err) {
            chan->in_use = 0;
            chan->irq_claimed = 0;
            local_irq_restore(flags);

            printk(KERN_ERR "%s: cannot get IRQ %d for DMA %d/n",
                   client->name, chan->irq, chan->number);
            return err;
        }

        chan->irq_enabled = 1;
    }

    local_irq_restore(flags);

    /* need to setup */

    pr_debug("%s: channel initialised, %p/n", __func__, chan);

    return chan->number | DMACH_LOW_LEVEL;
}
```

下面的函数是找通道好，先在板子通道映射中找，再在芯片通道映射中找。

```c
static struct s3c2410_dma_chan *s3c2410_dma_map_channel(int channel)
{
    struct s3c24xx_dma_order_ch *ord = NULL;
    struct s3c24xx_dma_map *ch_map;
    struct s3c2410_dma_chan *dmach;
    int ch;

    if (dma_sel.map == NULL || channel > dma_sel.map_size)
        return NULL;
    /*获得芯片的虚拟通道与真实通道映射的结构*/
    ch_map = dma_sel.map + channel;

    /* first, try the board mapping */
    /*如果有板子通道映射*/
    if (dma_order) {
        /*得到对应虚拟通道的所有真实通道的结构*/
        ord = &dma_order->channels[channel];
        /*找这个虚拟通道对应的每一个真实通道，看有没有有效并且未被使用的*/
        for (ch = 0; ch < dma_channels; ch++) {
            if (!is_channel_valid(ord->list[ch]))
                continue;

            if (s3c2410_chans[ord->list[ch]].in_use == 0) {
                ch = ord->list[ch] & ~DMA_CH_VALID;
                goto found;
            }
        }

        if (ord->flags & DMA_CH_NEVER)
            return NULL;
    }
    /*检查芯片虚拟通道与真实通道的映射，看有没有有效且未被使用的真实通道*/
    for (ch = 0; ch < dma_channels; ch++) {
        if (!is_channel_valid(ch_map->channels[ch]))
            continue;

        if (s3c2410_chans[ch].in_use == 0) {
            printk("mapped channel %d to %d/n", channel, ch);
            break;
        }
    }
    if (ch >= dma_channels)
        return NULL;
    /* update our channel mapping */

 found:
    /*将找到的通道保存在 dmach 中，并返回*/
    dmach = &s3c2410_chans[ch];
    dmach->map = ch_map;
    dma_chan_map[channel] = dmach;

    /* select the channel */
    /*调用选择通道的函数*/
    (dma_sel.select)(dmach, ch_map);

    return dmach;
}
```

设置寄存器，设置寄存器的工作由 s3c2410_dma_devconfig 和 s3c2410_dma_config 完成：

```
int s3c2410_dma_devconfig(int channel,
            enum s3c2410_dmasrc source,
            int hwcfg,
            unsigned long devaddr)
{
 /*根据虚拟通道号找到真实通道*/
 struct s3c2410_dma_chan *chan = lookup_dma_channel(channel);

 if (chan == NULL)
     return -EINVAL;

 pr_debug("%s: source=%d, hwcfg=%08x, devaddr=%08lx/n",
     __func__, (int)source, hwcfg, devaddr);

 chan->source = source;     //保存 dma 源
 chan->dev_addr = devaddr;  //保存 dma 源地址
 chan->hw_cfg = hwcfg;      //保存 dma 源的控制信息

 switch (source) {
 case S3C2410_DMASRC_HW:                    //源是外设，从外设读数据到内存，源的地址是固定的
     /* source is hardware */
     pr_debug("%s: hw source, devaddr=%08lx, hwcfg=%d/n",
         __func__, devaddr, hwcfg);
     dma_wrreg(chan, S3C2410_DMA_DISRCC, hwcfg & 3);         //初始化源控制寄存器
     dma_wrreg(chan, S3C2410_DMA_DISRC,  devaddr);           //将源地址写入初始源寄存器
     dma_wrreg(chan, S3C2410_DMA_DIDSTC, (0<<1) | (0<<0));   //目的地在 AHB 总线上

     chan->addr_reg = dma_regaddr(chan, S3C2410_DMA_DIDST);
     break;

 case S3C2410_DMASRC_MEM:                   //源是内存，从内存读数据到外设上，目的地址是固定的
     /* source is memory */
     pr_debug("%s: mem source, devaddr=%08lx, hwcfg=%d/n",
         __func__, devaddr, hwcfg);
     dma_wrreg(chan, S3C2410_DMA_DISRCC, (0<<1) | (0<<0));   //目的地址在 AHB 总线上
     dma_wrreg(chan, S3C2410_DMA_DIDST,  devaddr);           //把目的地址写到初始目的寄存器
     dma_wrreg(chan, S3C2410_DMA_DIDSTC, hwcfg & 3);         //初始化目的的控制寄存器

     chan->addr_reg = dma_regaddr(chan, S3C2410_DMA_DISRC);
     break;
     /*无论内存是源还是目的，这个地址始终是保存在 chan->addr_reg*/
 default:
     printk(KERN_ERR "dma%d: invalid source type (%d)/n",
            channel, source);

     return -EINVAL;
```

```
    }

    if (dma_sel.direction != NULL)
        (dma_sel.direction)(chan, chan->map, source);

    return 0;
}
```

```
int s3c2410_dma_config(unsigned int channel,
               int xferunit,
               int dcon)
{
    /*找到虚拟通道对应的实际通道*/
    struct s3c2410_dma_chan *chan = lookup_dma_channel(channel);

    pr_debug("%s: chan=%d, xfer_unit=%d, dcon=%08x/n",
         __func__, channel, xferunit, dcon);

    if (chan == NULL)
        return -EINVAL;

    pr_debug("%s: Initial dcon is %08x/n", __func__, dcon);
    /*清除 DMA 源的选择位*/
    dcon |= chan->dcon & dma_sel.dcon_mask;

    pr_debug("%s: New dcon is %08x/n", __func__, dcon);
    /*传输数据的大小*/
    switch (xferunit) {
    case 1:
        dcon |= S3C2410_DCON_BYTE;
        break;

    case 2:
        dcon |= S3C2410_DCON_HALFWORD;
        break;

    case 4:
        dcon |= S3C2410_DCON_WORD;
        break;

    default:
        pr_debug("%s: bad transfer size %d/n", __func__, xferunit);
        return -EINVAL;
    }

    dcon |= S3C2410_DCON_HWTRIG;     //DMA 源是硬件
    dcon |= S3C2410_DCON_INTREQ;     //中断使能

    pr_debug("%s: dcon now %08x/n", __func__, dcon);
    /*将通道控制寄存器和传输大小存于 chan 中*/
    chan->dcon = dcon;
    chan->xfer_unit = xferunit;

    return 0;
}
```

设置回调函数：

```
int s3c2410_dma_set_buffdone_fn(unsigned int channel, s3c2410_dma_cbfn_t rtn)
{
    。。。。。。。
    chan->callback_fn = rtn;

    return 0;
}
```

设置标志：

```
int s3c2410_dma_setflags(unsigned int channel, unsigned int flags)
{
    。。。。。。。。。。。。。。
    chan->flags = flags;

    return 0;
}
```

将数据放入队列，先看一下一个结构：

```
struct s3c2410_dma_buf {
        struct s3c2410_dma_buf    *next;
        int                  magic;         /* magic */
        int                  size;          /* buffer size in bytes */
        dma_addr_t           data;          /* start of DMA data */
        dma_addr_t           ptr;           /* where the DMA got to [1] */
        void                 *id;       /* client's id */
};
```

每个 struct s3c2410_dma_chan 维护了一个缓冲区队列，每个缓冲区用上边的结构表示。在 struct s3c2410_dma_chan 中的结构是：

```
/* buffer list and information */
```

```
struct s3c2410_dma_buf    *curr;          /* current dma buffer */
struct s3c2410_dma_buf    *next;          /* next buffer to load */
struct s3c2410_dma_buf    *end;           /* end of queue */
```

下边这个函数就是完成将 s3c2410_dma_buf 放入这个队列中排队：

```
int s3c2410_dma_enqueue(unsigned int channel, void *id,
            dma_addr_t data, int size)
{
    /*找到虚拟通道对应的实际通道*/
    struct s3c2410_dma_chan *chan = lookup_dma_channel(channel);
    struct s3c2410_dma_buf *buf;
    unsigned long flags;

    if (chan == NULL)
        return -EINVAL;

    pr_debug("%s: id=%p, data=%08x, size=%d/n",
         __func__, id, (unsigned int)data, size);
    /*分配 s3c2410_dma_chan 结构的 buffer*/
    buf = kmem_cache_alloc(dma_kmem, GFP_ATOMIC);
    if (buf == NULL) {
        pr_debug("%s: out of memory (%ld alloc)/n",
             __func__, (long)sizeof(*buf));
        return -ENOMEM;
    }

    //pr_debug("%s: new buffer %p/n", __func__, buf);
    //dbg_showchan(chan);
    /*设置这个 buffer*/
    buf->next  = NULL;
    buf->data  = buf->ptr = data;  //指向要传输数据的地址
    buf->size  = size;             //该段 buffer 的大小
    buf->id    = id;
    buf->magic = BUF_MAGIC;

    local_irq_save(flags);
    /*加载的是该通道的第一段 buf*/
    if (chan->curr == NULL) {
        /* we've got nothing loaded... */
        pr_debug("%s: buffer %p queued onto empty channel/n",
             __func__, buf);

        chan->curr = buf;   //curr 指向现在生成的 buf
        chan->end  = buf;
        chan->next = NULL;
    } else {
        pr_debug("dma%d: %s: buffer %p queued onto non-empty channel/n",
             chan->number, __func__, buf);

        if (chan->end == NULL)
            pr_debug("dma%d: %s: %p not empty, and chan->end==NULL?/n",
                 chan->number, __func__, chan);
        /*从链表尾加入链表*/
        chan->end->next = buf;
        chan->end = buf;
    }

    /* if necessary, update the next buffer field */
    if (chan->next == NULL)
        chan->next = buf;

    if (chan->state == S3C2410_DMA_RUNNING) {                              //该 channel 正在运行
        if (chan->load_state == S3C2410_DMALOAD_1LOADED && 1) {        //已有 buf load 了
            if (s3c2410_dma_waitforload(chan, __LINE__) == 0) {     //等待 load
                printk(KERN_ERR "dma%d: loadbuffer:"
                       "timeout loading buffer/n",
                       chan->number);
                dbg_showchan(chan);
                local_irq_restore(flags);
                return -EINVAL;
            }
        }

        while (s3c2410_dma_canload(chan) && chan->next != NULL) {        //检查能否 load
            s3c2410_dma_loadbuffer(chan, chan->next);                  //load buffer
        }
    } else if (chan->state == S3C2410_DMA_IDLE) {                          //该 channel 空闲着
        if (chan->flags & S3C2410_DMAF_AUTOSTART) {                     //如果设了自动启动标记，则直接启动该次传输
            s3c2410_dma_ctrl(chan->number | DMACH_LOW_LEVEL,          //启动传输
                     S3C2410_DMAOP_START);
        }
    }

    local_irq_restore(flags);
    return 0;
}
```

channel 在运行的时候会有很多状态，在 arch/arm/mach-s3c2410/include/mach/dma.h，注意已经很清楚了，我就不多解释了。

```
/* enum s3c2410_dma_loadst
 *
 * This represents the state of the DMA engine, wrt to the loaded / running
 * transfers. Since we don't have any way of knowing exactly the state of
 * the DMA transfers, we need to know the state to make decisions on wether
 * we can
 *
 * S3C2410_DMA_NONE
 *
 * There are no buffers loaded (the channel should be inactive)
 *
 * S3C2410_DMA_1LOADED
 *
 * There is one buffer loaded, however it has not been confirmed to be
 * loaded by the DMA engine. This may be because the channel is not
 * yet running, or the DMA driver decided that it was too costly to
 * sit and wait for it to happen.
 *
 * S3C2410_DMA_1RUNNING
 *
 * The buffer has been confirmed running, and not finisged
 *
 * S3C2410_DMA_1LOADED_1RUNNING
 *
 * There is a buffer waiting to be loaded by the DMA engine, and one
 * currently running.
*/

enum s3c2410_dma_loadst {
    S3C2410_DMALOAD_NONE,
    S3C2410_DMALOAD_1LOADED,
    S3C2410_DMALOAD_1RUNNING,
    S3C2410_DMALOAD_1LOADED_1RUNNING,
};
```

中断处理函数：

```
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;                           //当前传输完毕的 buf

    dbg_showchan(chan);
```

```c

/* modify the channel state */

switch (chan->load_state) {                    //改变状态，如果对上边那 4 个状态理解了很容易看懂的
case S3C2410_DMALOAD_1RUNNING:
    /* TODO - if we are running only one buffer, we probably
     * want to reload here, and then worry about the buffer
     * callback */

    chan->load_state = S3C2410_DMALOAD_NONE;
    break;

case S3C2410_DMALOAD_1LOADED:
    /* iirc, we should go back to NONE loaded here, we
     * had a buffer, and it was never verified as being
     * loaded.
     */

    chan->load_state = S3C2410_DMALOAD_NONE;
    break;

case S3C2410_DMALOAD_1LOADED_1RUNNING:
    /* we'll worry about checking to see if another buffer is
     * ready after we've called back the owner. This should
     * ensure we do not wait around too long for the DMA
     * engine to start the next transfer
     */

    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;

default:
    printk(KERN_ERR "dma%d: IRQ in invalid load_state %d/n",
           chan->number, chan->load_state);
    break;
}

if (buf != NULL) {                                    //如果不为空
    /* update the chain to make sure that if we load any more
     * buffers when we call the callback function, things should
```

```c
         * work properly */

        chan->curr = buf->next;                         //指向下一个 buf
        buf->next  = NULL;

        if (buf->magic != BUF_MAGIC) {
            printk(KERN_ERR "dma%d: %s: buf %p incorrect magic/n",
                   chan->number, __func__, buf);
            return IRQ_HANDLED;
        }

        s3c2410_dma_buffdone(chan, buf, S3C2410_RES_OK);          //buf 传输完成后的操作

        /* free resouces */
        s3c2410_dma_freebuf(buf);                         //释放 buf
    } else {
    }

    /* only reload if the channel is still running... our buffer done
     * routine may have altered the state by requesting the dma channel
     * to stop or shutdown... */

    /* todo: check that when the channel is shut-down from inside this
     * function, we cope with unsetting reload, etc */

    if (chan->next != NULL && chan->state != S3C2410_DMA_IDLE) {          //还有要传输的 buf,则继续传输
        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:
            if (s3c2410_dma_waitforload(chan, __LINE__) == 0) {     //如果已经有载入的，则等待被载入
                /* flag error? */
                printk(KERN_ERR "dma%d: timeout waiting for load (%s)/n",
                       chan->number, __func__);
                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);
    s3c2410_dma_loadbuffer(chan, chan->next);                    //载入 buf
    local_irq_restore(flags);
  } else {                                                  //所有传输完成
    s3c2410_dma_lastxfer(chan);                                //完成处理工作

    /* see if we can stop this channel.. */
    if (chan->load_state == S3C2410_DMALOAD_NONE) {
        pr_debug("dma%d: end of transfer, stopping channel (%ld)/n",
             chan->number, jiffies);
        s3c2410_dma_ctrl(chan->number | DMACH_LOW_LEVEL,          //停止 DMA 传输
                 S3C2410_DMAOP_STOP);
    }
  }

no_load:
  return IRQ_HANDLED;
}
```

可以选择不同的 dma 操作：

```
int
s3c2410_dma_ctrl(unsigned int channel, enum s3c2410_chan_op op)
{
  struct s3c2410_dma_chan *chan = lookup_dma_channel(channel);

  if (chan == NULL)
    return -EINVAL;

  switch (op) {
   case S3C2410_DMAOP_START:
     return s3c2410_dma_start(chan);

   case S3C2410_DMAOP_STOP:
```

```
		return s3c2410_dma_dostop(chan);

	case S3C2410_DMAOP_PAUSE:
	case S3C2410_DMAOP_RESUME:
		return -ENOENT;

	case S3C2410_DMAOP_FLUSH:
		return s3c2410_dma_flush(chan);

	case S3C2410_DMAOP_STARTED:
		return s3c2410_dma_started(chan);

	case S3C2410_DMAOP_TIMEOUT:
		return 0;
	}
	return -ENOENT;      /* unknown, don't bother */
}
```

分享到：