The current cdma driver does not support DMA_SLAVE_CONFIG !

也就是说cdma driver不支持 dmaengine_slave_config() API.

```
static inline int dmaengine_slave_config(struct dma_chan *chan,
                                          struct dma_slave_config *config)
{
        return dmaengine_device_control(chan,DMA_SLAVE_CONFIG,
                        (unsigned long)config);
}
```

According to Documentation/dmaengine.txt

The slave DMA usage consists of following steps:

1. Allocate a DMA slave channel

2. Set slave and controller specific parameters

3. Get a descriptor for transaction

4. Submit the transaction

5. Issue pending requests and wait for callback notification

Step 2 is "Set slave and controller specific parameters".

由Lexmark developer开发的cdma driver没有按照常规思路（也就是上面的思路）来，而是在step 1

(Allocate a DMA slave channel)中完成了config setting，而在device io control interface

中反而没有支持dmaengine_slave_config.

对stepper_cdma.c分析如下：

```
stmotors[motor_id].stmotor_dma_chan = dma_request_channel(mask,filter, &(stmotors
[motor_id].tx_dma_slave));
```

这里对stmotors[motor_id].tx_dma_slave的初始化如下

```c
static void smot_step_set_blk_addr_irq(stmotor_id_t motor_id, uint32_t cdma_num)
{
        stmotors[motor_id].tx_dma_slave.vtype = MV61_VDMA_OWNED;
        stmotors[motor_id].tx_dma_slave.wr_delay = 0;
        stmotors[motor_id].tx_dma_slave.destendian = MV61_DMA_LITTLE_ENDIAN;
        stmotors[motor_id].tx_dma_slave.srcendian = MV61_DMA_LITTLE_ENDIAN;
        stmotors[motor_id].tx_dma_slave.flowctrl = MV61_DMA_MEMORY_TO_PERIPHERAL;
        stmotors[motor_id].tx_dma_slave.dest_pid = cdma_num;
        stmotors[motor_id].tx_dma_slave.dest_addr_inc = true;
        stmotors[motor_id].tx_dma_slave.src_addr_inc = true;
        stmotors[motor_id].tx_dma_slave.dest_width = MV61_DMA_XFER_WIDTH_32BIT;
        stmotors[motor_id].tx_dma_slave.src_width = MV61_DMA_XFER_WIDTH_32BIT;
        stmotors[motor_id].tx_dma_slave.data_unit_size = MV61_DMA_UNIT_SIZE_32BIT;
        stmotors[motor_id].tx_dma_slave.dest_burst = MV61_DMA_BURST1;
        stmotors[motor_id].tx_dma_slave.src_burst = MV61_DMA_BURST1;
        stmotors[motor_id].tx_dma_slave.dest_reg = (dma_addr_t)&(stmotors[motor_id].phy_stmotor_regs->PWM_T);
        stmotors[motor_id].tx_dma_slave.timebase = MV61_TIMEBASE_1MS;
        stmotors[motor_id].tx_dma_slave.timer = 0;
        stmotors[motor_id].tx_dma_slave.wrap = 24;

}
```

in include/linux/platform_data/mv61_cdma.h

```c
/**
 * filter - filter candidate dma channels and initialize if ok
 * @chan: channel candidate offered by dmaengine's dma_request_channel()
 * @slave: contains requested virtual channel type and peripheral data
 */
static bool __attribute__((used)) filter(struct dma_chan *chan, void *slave) {
        struct mv61_dma_slave *vslave = slave;

        if (__chk_vdma_type(chan, vslave->vtype)) {
                chan->private = vslave;
                return true;
        } else
                return false;
}
```

在filter()中会把stmotors[motor_id].tx_dma_slave的address assign to chan->private。

__dma_request_channel()

{

    ......

    chan = private_candidate(mask, device, fn, fn_param);

    ......

    err = dma_chan_get(chan);

    ......

```
}
```

private_candidate() invokes filter callback, make chan->private = fn_param.

```c
static int dma_chan_get(struct dma_chan *chan)
{
        int err = -ENODEV;
        struct module *owner = dma_chan_to_owner(chan);

        if (chan->client_count) {
                __module_get(owner);
                err = 0;
        } else if (try_module_get(owner))
                err = 0;

        if (err == 0)
                chan->client_count++;

        /* allocate upon first client reference */
        if (chan->client_count == 1 && err == 0) {
                int desc_cnt = chan->device->device_alloc_chan_resources(chan);

                if (desc_cnt < 0) {
                        err = desc_cnt;
                        chan->client_count = 0;
                        module_put(owner);
                } else if (!dma_has_cap(DMA_PRIVATE, chan->device->cap_mask))
                        balance_ref_count(chan);
        }

        return err;
}
```

调用mv61vc_alloc_chan_resources()。

```c
static int mv61vc_alloc_chan_resources(struct dma_chan *chan)
{
        struct mv61_vdma_chan   *mv61vc = dchan_to_mv61_vdma_chan(chan);
        struct mv61_dma_slave   *mv61s;
        unsigned long           lockvcflags;

        __dev_vdbg(chan2dev(chan), "alloc_chan_resources\n");

        spin_lock_irqsave(&mv61vc->lock, lockvcflags);

        /* ASSERT:  channel is idle */
        BUG_ON(!list_empty(&mv61vc->queue));
        BUG_ON(!list_empty(&mv61vc->active_list));
        BUG_ON(!list_empty(&mv61vc->complete_list));

        mv61vc->completed = chan->cookie = 1;

        mv61s = chan->private;
        if (mv61s) {
                /*
                 * We need controller-specific data to set up slave
                 * transfers.
                 */
                mv61vc->wrap = mv61s->wrap;
                mv61vc->def.valid = 1;
                mv61vc->def.TimerControl = 0;
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.SrcAddr, CDMA_SRCADDR_SRCADDR,
                                                                mv61s->src_reg);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.DestAddr, CDMA_DESTADDR_DESTADDR,
                                                                mv61s->dest_reg);
                if(mv61s->timer) {

                        MV61_CDMA_VAR_WR_FIELD(mv61vc->def.TimerControl,
                                        CDMA_TIMERCONTROL_TIMERENABLE, 1);
                        MV61_CDMA_VAR_WR_FIELD(mv61vc->def.TimerControl,
                                        CDMA_TIMERCONTROL_TIMEBASE,
                                        mv61s->timebase);
                        MV61_CDMA_VAR_WR_FIELD(mv61vc->def.TimerControl,
                                        CDMA_TIMERCONTROL_COUNT,
                                        mv61s->timer);
                }

                mv61vc->def.Control = 0;
                mv61vc->def.CFG = 0;
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_WRDELAY,
                                                        mv61s->wr_delay);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_DATAUNITSIZE,
                                                        mv61s->data_unit_size);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_FLOWCTRL,
                                                        mv61s->flowctrl);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_DESTPID,
                                                        mv61s->dest_pid);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_SRCPID,
                                                        mv61s->src_pid);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_DESTENDIAN,
                                                        mv61s->destendian);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.CFG,CDMA_CFG_SRCENDIAN,
                                                        mv61s->srcendian);

                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.Control,CDMA_CONTROL_DESTADDRINC,
                                                        mv61s->dest_addr_inc);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.Control,CDMA_CONTROL_SRCADDRINC,
                                                mv61s->src_addr_inc);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.Control,CDMA_CONTROL_DESTXFERWIDTH,
                                                mv61s->dest_width);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.Control,CDMA_CONTROL_SRCXFERWIDTH,
                                                mv61s->src_width);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.Control,CDMA_CONTROL_DESTBURSTSIZE,
                                                mv61s->dest_burst);
                MV61_CDMA_VAR_WR_FIELD(mv61vc->def.Control,CDMA_CONTROL_SRCBURSTSIZE,
                                                mv61s->src_burst);
 =      }

        spin_unlock_irqrestore(&mv61vc->lock, lockvcflags);
        return 0;
}
```

mv61s = chan->private;

cdma driver code(cdma_main.c)获得了stepper_cdma.c中的对cdma的setting

然后是把stepper_cdma中的setting设置到virtual channel的def field.

```
struct mv61_vdma_chan {
        struct dma_chan         chan;
        struct mv61_vdma        *mv61v;

        struct mv61_vreg_defs   def;
        int                     wrap;

        spinlock_t              lock;

        /* these elements are all protected by lock */
        u32                     irqs;
        dma_cookie_t            completed;
        dma_cookie_t            started;
        struct list_head        active_list;
        struct list_head        complete_list;
        struct list_head        queue;
        struct mv61_stat_regs   hwstat;
        int                     residue;
        enum dma_status         status;

};
```

这还只是设置在software中，并没有设置到cdma的control register。

```
struct mv61_vreg_defs {
        bool valid;
        u32 CFG;
        u32 Control;
        dma_addr_t SrcAddr;
        dma_addr_t DestAddr;
        u32 TimerControl;
};
```

```
 * @CFG: DMA Configuration Register
 * @Control: DMA Channel Control Register
 * @SrcAddr: DMA Source Address Register
 * @DestAddr: DMA Destination Address Register
 * @TimerControl: DMA Timer and Timebase Control
```

而真正往cdma registers中写是在mv61vc_dostart()中。

```c
int mv61vc_dostart(struct mv61_vdma_chan *mv61vc, struct mv61_desc *desc)
{
        struct mv61_pdma_chan    *mv61pc;
        struct mv61_pdma_chan_regs *pcregs;
        u32                     tmp;

        mv61pc = mv61_vpmap_v_to_p(mv61vc);
        BUG_ON(!mv61pc);

        pcregs = mv61pc->ch_regs;                ①

        /* ASSERT:  channel is idle */
        tmp = readl(&pcregs->CFG);
        if (tmp & CDMA_CFG_ENABLE_MASK) {
                __dev_vdbg(chan2dev(&mv61vc->chan), "dostart:%d\n", __LINE__);
                /* The tasklet will hopefully advance the queue... */
                return -EBUSY;
        }

        memset(&mv61vc->hwstat, 0, sizeof(mv61vc->hwstat));

        mv61vc->residue = 0;
        mv61vc->status = DMA_IN_PROGRESS;

        writel(desc->txregs.CFG, &pcregs->CFG);                    ②
        writel(desc->txregs.FillValue, &pcregs->FillValue);
        writel(desc->txregs.intEn, &pcregs->intEn);
        writel(desc->txregs.TimerControl, &pcregs->TimerControl);

        writel(mv61vc->def.Control,&pcregs->Control );          ③

        wmb();

        tmp = readl(&pcregs->CFG);
        writel(tmp | CDMA_CFG_ENABLE_MASK, &pcregs->CFG);

        dma_sync_single_for_device(chan2parent(&mv61vc->chan), desc->lli_phys,
                        sizeof(desc->lli), DMA_TO_DEVICE);
        writel(desc->lli_phys,&pcregs->CDR);

        return 0;
}
```

cdma hardware registers的setting来自两处

1. mv61_desc

2. 来自cdma client的设置

①

pcregs 指向hardware register

②

来自每次传输的descriptor

③

来自cdma client的设置