Project 5 Device Driver 设计文档

中国科学院大学
陈灿宇
2019.1.1

1　网卡驱动

(1) 发送描述符初始化

由于我在本次实验中直接实现的 TASK3, 所以在正常的网络传输处理中, 初始化了 64 个发送描述符形成环形链表.

第 1-63 个描述符初始化的位域如下:

- des0：0x0
- des1：((1 << 30) 当前 buffer 包含的是一帧数据的最后一段 |(1 << 29) 当前 buffer 包含的是一帧数据的第一段 |(1 << 24) 环形链表的非最后一个描述符 | (bufsize & 0x7ff) 缓冲区大小)
- des2：发送缓冲区物理地址
- des3：下一个描述符的物理地址

最后一个描述符初始化的位域如下:

- des0：0x0
- des1：((1 << 30) 当前 buffer 包含的是一帧数据的最后一段 |(1 << 29) 当前 buffer 包含的是一帧数据的第一段 |(1 << 25) 环形链表的最后一个描述符 | (bufsize & 0x7ff) 缓冲区大小)
- des2：发送缓冲区物理地址
- des3：第一个描述符的物理地址

```
uint32_t do_send_desc_init(void *desc_addr, void *buffer, \
                           uint32_t bufsize, uint32_t pnum)
{
    int cnt = 0;
    uint32_t start_addr = (uint32_t)desc_addr;
    uint32_t addr = (uint32_t)desc_addr;

    //Not the last one
    while (++cnt < (pnum))
    {
        ((desc_t *)addr)->des0 = 0x00000000;
        ((desc_t *)addr)->des1 = (0|(0<<31)|(1<<30)|(1<<29)|(1<<24)|(bufsize&0x7ff));
        ((desc_t *)addr)->des2 = ((uint32_t)buffer) & 0x1fffffff;
        ((desc_t *)addr)->des3 = (addr + DESC_SIZE) & 0x1fffffff;
```

```
15          addr += DESC_SIZE;
16      }
17
18      //The last one
19      ((desc_t *)addr)->des0 = 0x00000000;
20      ((desc_t *)addr)->des1 = (0|(0<<31)|(1<<30)|(1<<29)|(1<<25)|(1<<24)|(bufsize&0x7ff));
21      ((desc_t *)addr)->des2 = ((uint32_t)buffer) & 0x1fffffff;
22      ((desc_t *)addr)->des3 = (start_addr) & 0x1fffffff;
23      addr += DESC_SIZE;
24
25      return start_addr;
26  }
```

（2）接收描述符初始化

初始化了 64 个接收描述符形成环形链表.
第 1-63 个描述符初始化的位域如下:

- des0：0x0
- des1：((1 << 24) 环形链表的非最后一个描述符 |(1 << 31) 不触发网卡中断 | (bufsize & 0x7ff) 缓冲区大小)
- des2：接收缓冲区物理地址
- des3：下一个描述符的物理地址

最后一个描述符初始化的位域如下:

- des0：0x0
- des1：((1 << 25) 环形链表的最后一个描述符 |(0 << 31) 接收完成触发网卡中断 | (bufsize & 0x7ff) 缓冲区大小)
- des2：接收缓冲区物理地址
- des3：第一个描述符的物理地址

```
1   uint32_t do_recv_desc_init(void *desc_addr, void *buffer, \
2                              uint32_t bufsize, uint32_t pnum)
3   {
4       int cnt = 0;
5       uint32_t start_addr = (uint32_t)desc_addr;
6       uint32_t addr = (uint32_t)desc_addr;
7
8       //Not the last one
9       while (++cnt < (pnum))
10      {
11          ((desc_t *)addr)->des0 = 0x00000000;
```

```
        ((desc_t *)addr)->des1 = 0|(1<<24)|(1<<31)|(bufsize&0x7ff);
        ((desc_t *)addr)->des2 = ((uint32_t)buffer + (cnt - 1) * bufsize)& 0x1fffffff;
        ((desc_t *)addr)->des3 = (addr + DESC_SIZE)& 0x1fffffff;
        addr += DESC_SIZE;
    }

    //The last one
    ((desc_t *)addr)->des0 = 0x00000000;
    ((desc_t *)addr)->des1 = 0|(1<<25)|(0<<31)|(bufsize&0x7ff);
    ((desc_t *)addr)->des2 = ((uint32_t)buffer + (cnt - 1) * bufsize)& 0x1fffffff;
    ((desc_t *)addr)->des3 = (start_addr)& 0x1fffffff;
    addr += DESC_SIZE;

    return start_addr;
}
```

（3）任务二实现时，检查是否有数据包到达网卡这一操作是在哪个流程中执行的？例如是时钟处理流程，还是接收线程本身，或者是其他流程中？

在时钟处理流程中。

（4）网卡中断设计

在原有的中断处理路径上添加了简单的判断. 将网卡中断的相关处理直接固定在中断处理流程中. 通过读取 CP0_CAUSE 寄存器和 INT1_SR 寄存器来实现对网卡中断的判断.

```
NESTED(handle_int, 0, sp)
...
    mfc0  k0, CP0_CAUSE
    nop
    andi  k0, k0, CAUSE_IPL
    clz   k0, k0
    nop
    xori  k0, k0, 0x17
    addiu k1, zero, 7
    beq   k0, k1, irq_timer
    nop
    jal   check_irq_mac
    nop
    jal   clear_int
    nop
...
END(handle_int)

void check_irq_mac()
{
    uint32_t CP0_CAUSE = get_cp0_status();
    uint32_t ip = ((CP0_CAUSE >> 8) & 0x000000FF);
```

```
23
24      if(ip & (1<<3))//ip[3]==1
25      {
26          if((*(uint32_t*)INT1_SR)&(0x00000001<<3))
27          {
28              irq_mac();
29          }
30          return;
31      }
32  }
```

（5）DMA 接收和发送描述符采用环形链表和链型链表都是可以的，你认为使用环形链表和使用链型链表有什么区别？

此实验中使用环形链表. 一方面, 环形链表作为传输描述符在各种系统中应用的比较广泛. 另一方面, 使用环形链表可以使得 DMA 当前描述符寄存器在完成一组描述符之后返回到起始地址, 方便重复利用发送描述符.

（6）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

2 Bonus 设计

（1）相比较任务三而言，在 Bonus 中你是否有新增设计，以满足 Bonus 对网卡接收性能的要求？若有，请说明你的新增设计和用途。

Bonus 部分我只进行了初步设计，如果后续有时间会进一步进行完善。

初步的想法是增大描接收述符数量和缓冲区大小，经过测试，在将接收描述符增大到 256 个时，对于接收的速度有一定提高，可以达到 0.9Mbit, 但接收速度还未达到 1Mbit+。

3　关键函数功能

请列出上述各项功能设计里，你觉得关键的函数或代码块，及其作用

部分 DMA 寄存器配置需要按照顺序进行，否则会导致模拟器通过上板不通过，所以下面两个函数较为关键。

```
uint32_t do_net_recv(uint32_t rd, uint32_t rd_phy, uint32_t daddr)
{
    reg_write_32(DMA_BASE_ADDR + 0xC, (uint32_t)recv_desc_table_ptr);
    reg_write_32(GMAC_BASE_ADDR, reg_read_32(GMAC_BASE_ADDR) | 0x4);
    reg_write_32(DMA_BASE_ADDR + 0x18, reg_read_32(DMA_BASE_ADDR + 0x18) | 0x02200002);
    reg_write_32(DMA_BASE_ADDR + 0x1c, 0x10001 | (1 << 6));

    int l;
    for(l=0;l<64;l++)
    {
        ((desc_t*)(rd+l*DESC_SIZE))->des0=0x80000000;
    }

    for(l=0;l<64;l++)
    {
        reg_write_32(DMA_BASE_ADDR + DmaRxPollDemand, 0x1);
    }

    return 0; //if recev succeed
}
```

```
void do_net_send(uint32_t td, uint32_t td_phy)
{
    reg_write_32(DMA_BASE_ADDR + 0x10, td_phy);
    reg_write_32(GMAC_BASE_ADDR, reg_read_32(GMAC_BASE_ADDR) | (1 << 2) | (1 << 3) );
    reg_write_32(DMA_BASE_ADDR + 0x18, reg_read_32(DMA_BASE_ADDR + 0x18) | 0x02202000);
    reg_write_32(DMA_BASE_ADDR + 0x1c, 0x10001 | (1 << 6));

    int l;
    for(l=0;l<64;l++)
    {
        ((desc_t*)(td+l*DESC_SIZE))->des0=0x80000000;
    }

    for(l=0;l<64;l++)
    {
        reg_write_32(DMA_BASE_ADDR + DmaTxPollDemand, 0x1);
    }

    return;
}
```