Project 5 Device Driver 设计文档

中国科学院大学
[薛峰]
[2018/12/23]

1. 网卡驱动

（1）任务一实现时，你初始化了几个接收描述符（RDES），每个接收描述符中设置了哪几个域的值，所设置的域的各自含义是什么？

答：任务一中初始化了 64 个接收描述符。

rdes0 的第 31 位即 OWN 域设置为 0，表明还未开始接收包；

rdes1 的第 24 位设置为 1，表明 buffer2 中的地址指向下一个接收描述符，低 11 位设置为 0x400 表示 buffer1 的大小为 1KB，其余位置设置为 0。另外对于最后一个描述符，第 25 位设置为 1，表明该描述符是环型描述符链表中的最后一个；

rdes2 设置为 buffer1 的物理地址；

rdes3 设置为下一个描述符的物理地址（最后一个要指向第一个）。

（2）任务一实现时，每个发送描述符中设置了哪几个域的值，所设置的域的各自含义是什么？

答：任务一中初始化了 64 个发送描述符。

rdes0 的第 31 位即 OWN 域设置为 0，表明还未开始发送包；

rdes1 的第 24 位设置为 1，表明 buffer2 中的地址指向下一个发送描述符，低 11 位设置为 0x400 表示 buffer1 的大小为 1KB，其余位置设置为 0。另外对于最后一个描述符，第 25 位设置为 1，表明该描述符是环型描述符链表中的最后一个；

rdes2 设置为 buffer1 的物理地址；

rdes3 设置为下一个描述符的物理地址（最后一个要指向第一个）。

（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？

答：在时钟中断处理流程中检查是否有接收到包。

（4）任务三实现时，你的设计中，每接收到几个网络包时会产生一次中断？

答：没接收到 64 个网络包时会产生一次中断。

（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？

答：环形链表可以重复利用描述符，从而能够用较少的描述符接收更多的包。

（6）设计或实现过程中遇到的问题和得到的经验。

答：在实现任务三的过程中，没有完全理解接受描述符 rdes1 第 31 位的作用，导致每个描述赋的 rdes1 第 31 位都置为 1，所以在上板验证的过程中已知没有触发网卡中断，并且检查了很长时间才发现这个错误。

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。

答：将接收描述符的个数从 64 个增加到了 256 个，从而能够实现在 10s 内接收到大约 20000 个包，从而实现了接收速度要到 1Mbit/s 以上。扩大接收描述符的个数，使得从原来的接收到 64 个包就发生网卡中断变为接收到 256 个包才发生网卡中断，从而减少了处理网卡中断的时间，从而提高了接收速度。

（2）设计或实现过程中遇到的问题和得到的经验。

答：更改接收描述符的个数需要改的地方比较多，稍微不注意就会漏掉某个地方。例如在做该任务的时候，对最后一个描述符进行初始化的过程中，忘记更改对 resd2 赋值语句等号右边的值，导致没有将第 255 个描述符的 rdes2 设置为第 255 个 buffer 的地址，而是设置为第 63 个 buffer 的地址，导致上板结果不正确。

3. 关键函数功能

一、void do_net_send(uint32_t td, uint32_t td_phy)：开始发送数据包

```
void do_net_send(uint32_t td, uint32_t td_phy)
{
    int i;
    desc_t *s_desc = (desc_t *)td;
    //PLEASE enable MAC-TX
    reg_write_32(DMA_BASE_ADDR + DmaTxBaseAddr, td_phy);                    // DMA #4  Physical head address of DMA
    reg_write_32(GMAC_BASE_ADDR, reg_read_32(GMAC_BASE_ADDR) | GmacTxEnable);   // enable send

    reg_write_32(DMA_BASE_ADDR + 0x18, reg_read_32(DMA_BASE_ADDR + 0x18) | 0x02202000); //0x02202002); // start tx, rx // DMA #6
    reg_write_32(DMA_BASE_ADDR + 0x1c, 0x10001 | (1 << 6));                                                  // DMA #7

    //you should add some code to start send packages
    for(i = 0; i < PNUM; i++)
        s_desc[i].tdes0 = DescOwnByDma;                  // enable send
    for(i = 0; i < PNUM; i++)
        reg_write_32(DMA_BASE_ADDR + DmaTxPollDemand, 1);  // # DMA 1    start send data
}
```

二、uint32_t do_net_recv(uint32_t rd, uint32_t rd_phy, uint32_t daddr)：开始接收数据包

```
uint32_t do_net_recv(uint32_t rd, uint32_t rd_phy, uint32_t daddr)
{
    int i, j;
    int count = 0;
    uint32_t *recv_addr = (uint32_t *)daddr;
    desc_t *r_desc = (desc_t *)rd;
    //PLEASE enable MAC-RX
    reg_write_32(DMA_BASE_ADDR + DmaRxBaseAddr, rd_phy);                    // DMA #3  Physical head address of DMA
    reg_write_32(GMAC_BASE_ADDR, reg_read_32(GMAC_BASE_ADDR) | GmacRxEnable);   // enable send

    reg_write_32(DMA_BASE_ADDR + 0x18, reg_read_32(DMA_BASE_ADDR + 0x18) | 0x02200002); // start tx, rx  // DMA #6
    reg_write_32(DMA_BASE_ADDR + 0x1c, 0x10001 | (1 << 6));                                               // DMA #7

    //you should add some code to start recv and check recv packages
    for(i = 0; i < PNUM; i++)
        r_desc[i].tdes0 = DescOwnByDma;                  // enable recv

    for(i = 0; i < PNUM; i++)
        reg_write_32(DMA_BASE_ADDR + DmaRxPollDemand, 1);  // # DMA 2    start receive data

    return 0;
}
```

三、void_mac(void)：唤醒接收数据包进程，并清除网卡中断。

```
void irq_mac(void)
{
    if(!queue_is_empty(&recv_block_queue))
        do_unblock_one(&recv_block_queue);
    clear_interrupt();
}
```

四、void send_desc_init(mac_t *mac)：初始化接收描述符。

```
static void send_desc_init(mac_t *mac)
{
    int i, j;

    for(i = 0; i < PNUM; i++)
        for(j = 0; j < PSIZE; j ++)
            Tx_buffer[i][j] = 0;

    for(i = 0; i < 63; i++)
    {
        Tx_desc[i].tdes0 = 0;
        Tx_desc[i].tdes1 = DescTxLast | DescTxFirst | TxDescChain | PSIZE*4;
        Tx_desc[i].tdes2 = ((int)Tx_buffer[i])   & 0x1fffffff;   // Physical Address
        Tx_desc[i].tdes3 = ((int) &Tx_desc[i+1]) & 0x1fffffff;   // Physical Address
    }

    Tx_desc[63].tdes0 = 0;
    Tx_desc[63].tdes1 = DescTxLast | DescTxFirst | TxDescEndOfRing |TxDescChain | PSIZE*4;
    Tx_desc[63].tdes2 = ((int)Tx_buffer[63]) & 0x1fffffff;
    Tx_desc[63].tdes3 = ((int) &Tx_desc[0])  & 0x1fffffff;

    mac->saddr = (uint32_t)Tx_buffer[0];
    mac->saddr_phy = ((uint32_t)Tx_buffer[0]) & 0x1fffffff;

    mac->td = (uint32_t)&Tx_desc[0];
    mac->td_phy = ((uint32_t)&Tx_desc[0]) & 0x1fffffff;

    for(i = 0; i < PNUM; i++)
        for(j = 0; j < PSIZE; j++)
            Tx_buffer[i][j] = buffer[j];
}
```

五、void_desc_init(mac_t *mac)：初始化发送描述符。

```
static void recv_desc_init(mac_t *mac)
{
    int i, j;

    for(i = 0; i < PNUM; i++)
        for(j = 0; j < PSIZE; j ++)
            Rx_buffer[i][j] = 0;

    for(i = 0; i < 63; i++)
    {
        Rx_desc[i].tdes0 = 0;
        Rx_desc[i].tdes1 = RxDisIntCompl | RxDescChain | PSIZE*4;
        Rx_desc[i].tdes2 = ((int)Rx_buffer[i])  & 0x1fffffff;   // Physical Address
        Rx_desc[i].tdes3 = ((int)&Rx_desc[i+1]) & 0x1fffffff;   // Physical Address
    }

    Rx_desc[63].tdes0 = 0;
    Rx_desc[63].tdes1 = /*RxDisIntCompl | */ RxDescEndOfRing | RxDescChain | PSIZE*4;
    Rx_desc[63].tdes2 = ((int)Rx_buffer[63]) & 0x1fffffff;
    Rx_desc[63].tdes3 = ((int)&Rx_desc[0])   & 0x1fffffff;

    mac->daddr = (uint32_t)Rx_buffer[0];
    mac->daddr_phy = ((uint32_t)Rx_buffer[0]) & 0x1fffffff;

    mac->rd = (uint32_t)&Rx_desc[0];
    mac->rd_phy = ((uint32_t)&Rx_desc[0]) & 0x1fffffff;
}
```