Project 5 Device Driver设计文档

中国科学院大学

蔡洛姗

2021/1/4

# 网卡驱动

（1）S-Core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？

描述符统一设置在bd\_space全局变量中，其中前32个用于接收，后32个用于发送。每个描述符分为两个word，对应位根据讲义的描述符位说明表来设置。

（2）在A-Core实现时，你在哪个函数中判断接收到的数据包数量已经足够了？另外，你的设计中，每接收到几个网络包时会产生一次中断？为什么这么设计？

在handle\_irq中通过扫描描述符的ownership位判断是否收到了足够多的包。中断信号会在收到不确定数量的包的时候到来，这个由网卡决定。

（3）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？

环形链表可以支持重复使用描述符，扩展性更强。

# C-Core设计

# 关键函数功能

**//系统调用**

**long** **do\_net\_recv**(**uintptr\_t** addr, **size\_t** length, **int** num\_packet, **size\_t**\* frLength)

{

**long** status;

**int** now\_packet = 0;

**uint8\_t** \*user\_addr = (**uint8\_t** \*)addr;

**while**(now\_packet<num\_packet){

**if**((num\_packet-now\_packet)<=TURN\_NUM){

status = EmacPsRecv(&EmacPsInstance, (EthernetFrame \*)rx\_buffers, (num\_packet-now\_packet));

**if**(net\_poll\_mode==1){ //task3:interrupt

**uint64\_t** cpu\_id;

cpu\_id = get\_current\_cpu\_id();

do\_block(&current\_running[cpu\_id]->list, &recv\_queue);

}

status = EmacPsWaitRecv(&EmacPsInstance, (num\_packet-now\_packet), rx\_len);

**for**(**int** i=0; i < (num\_packet-now\_packet); i++){

kmemcpy((**uint8\_t** \*)user\_addr, (**uint8\_t** \*)&rx\_buffers[i], rx\_len[i]);

user\_addr += rx\_len[i];

frLength[i+now\_packet] = rx\_len[i];

}

}**else**{

// set RX descripter and enable mac

status = EmacPsRecv(&EmacPsInstance, (EthernetFrame \*)rx\_buffers, TURN\_NUM);

// wait until you receive enough packets(`num\_packet`).

**if**(net\_poll\_mode==1){ //task3:interrupt

**uint64\_t** cpu\_id;

cpu\_id = get\_current\_cpu\_id();

do\_block(&current\_running[cpu\_id]->list, &recv\_queue);

}

status = EmacPsWaitRecv(&EmacPsInstance, TURN\_NUM, rx\_len);

// copy the buffer\_data and length\_array into user\_space

**for**(**int** i=0; i < TURN\_NUM; i++){

kmemcpy((**uint8\_t** \*)user\_addr, (**uint8\_t** \*)&rx\_buffers[i], rx\_len[i]);

user\_addr += rx\_len[i];

frLength[i+now\_packet] = rx\_len[i];

}

}

now\_packet += TURN\_NUM;

}

**return** status;

}

// send all packet (every call do\_net\_send() sends one packet)

**void** **do\_net\_send**(**uintptr\_t** addr, **size\_t** length)

{

kmemcpy((**uint8\_t** \*)tx\_buffer, (**uint8\_t** \*)addr, **sizeof**(EthernetFrame));

// set TX descripter and enable mac

**long** status;

status = EmacPsSend(&EmacPsInstance, (EthernetFrame \*)tx\_buffer, length);

// wait until dma finish sending a packet.

**if**(net\_poll\_mode==1){ //task3:interrupt

**uint64\_t** cpu\_id;

cpu\_id = get\_current\_cpu\_id();

do\_block(&current\_running[cpu\_id]->list, &send\_queue);

}

status = EmacPsWaitSend(&EmacPsInstance);

// maybe you need to call drivers' send function multiple times ?

}

**void** **do\_net\_irq\_mode**(**int** mode)

{

**if**(mode==1){

XEmacPs\_WriteReg(EmacPsInstance.Config.BaseAddress,XEMACPS\_IER\_OFFSET,XEMACPS\_IXR\_ALL\_MASK);

}

net\_poll\_mode = mode;

}

**//中断处理**

**void** **handle\_irq**(regs\_context\_t \*regs, **uint64\_t** irq)

{

// TODO:

// handle external irq from network device

**uint32\_t** ISR\_reg;

ISR\_reg = XEmacPs\_ReadReg(EmacPsInstance.Config.BaseAddress, XEMACPS\_ISR\_OFFSET);

**if**(ISR\_reg & XEMACPS\_IXR\_FRAMERX\_MASK){ //complete getting a packet

**int** recieved\_num=0;

**while**(bd\_space[recieved\_num]%2){

recieved\_num++;

}

**if**((bd\_space[recieved\_num-1]%4)==3){ //reieve enough packet

**if** (!list\_empty(&recv\_queue)) {

do\_unblock(recv\_queue.prev);

}

}

//set XEMACPS\_ISR\_OFFSET for complete recieve

XEmacPs\_WriteReg(EmacPsInstance.Config.BaseAddress, XEMACPS\_ISR\_OFFSET,ISR\_reg);

//set RXSR for complete recieve

XEmacPs\_WriteReg(EmacPsInstance.Config.BaseAddress, XEMACPS\_RXSR\_OFFSET,XEMACPS\_RXSR\_FRAMERX\_MASK);

// NOTE: remember to flush dcache

Xil\_DCacheFlushRange(0, 64);

// let PLIC know that handle\_irq has been finished

plic\_irq\_eoi(irq);

}

**if**(ISR\_reg & XEMACPS\_IXR\_TXCOMPL\_MASK){ //complete sending a packet

**if** (!list\_empty(&send\_queue)) {

do\_unblock(send\_queue.prev);

}

//set XEMACPS\_ISR\_OFFSET for complete recieve/send

XEmacPs\_WriteReg(EmacPsInstance.Config.BaseAddress, XEMACPS\_ISR\_OFFSET,ISR\_reg);

//set TXSR for complete recieve

**uint32\_t** TXSR\_reg;

TXSR\_reg = XEmacPs\_ReadReg(EmacPsInstance.Config.BaseAddress, XEMACPS\_TXSR\_OFFSET);

XEmacPs\_WriteReg(EmacPsInstance.Config.BaseAddress, XEMACPS\_TXSR\_OFFSET,TXSR\_reg|XEMACPS\_TXSR\_TXCOMPL\_MASK);

// NOTE: remember to flush dcache

Xil\_DCacheFlushRange(0, 64);

// let PLIC know that handle\_irq has been finished

plic\_irq\_eoi(irq);

}

}