浙江大学
ZHEJIANG UNIVERSITY

嵌入式系统

第12课 网络设备驱动程序设计

王总辉
zhwang@zju.edu.cn

http://course.zju.edu.cn

提纲

- 网络设备驱动程序设计概要
- 网络设备驱动程序设计方法
- 网络设备驱动程序简介

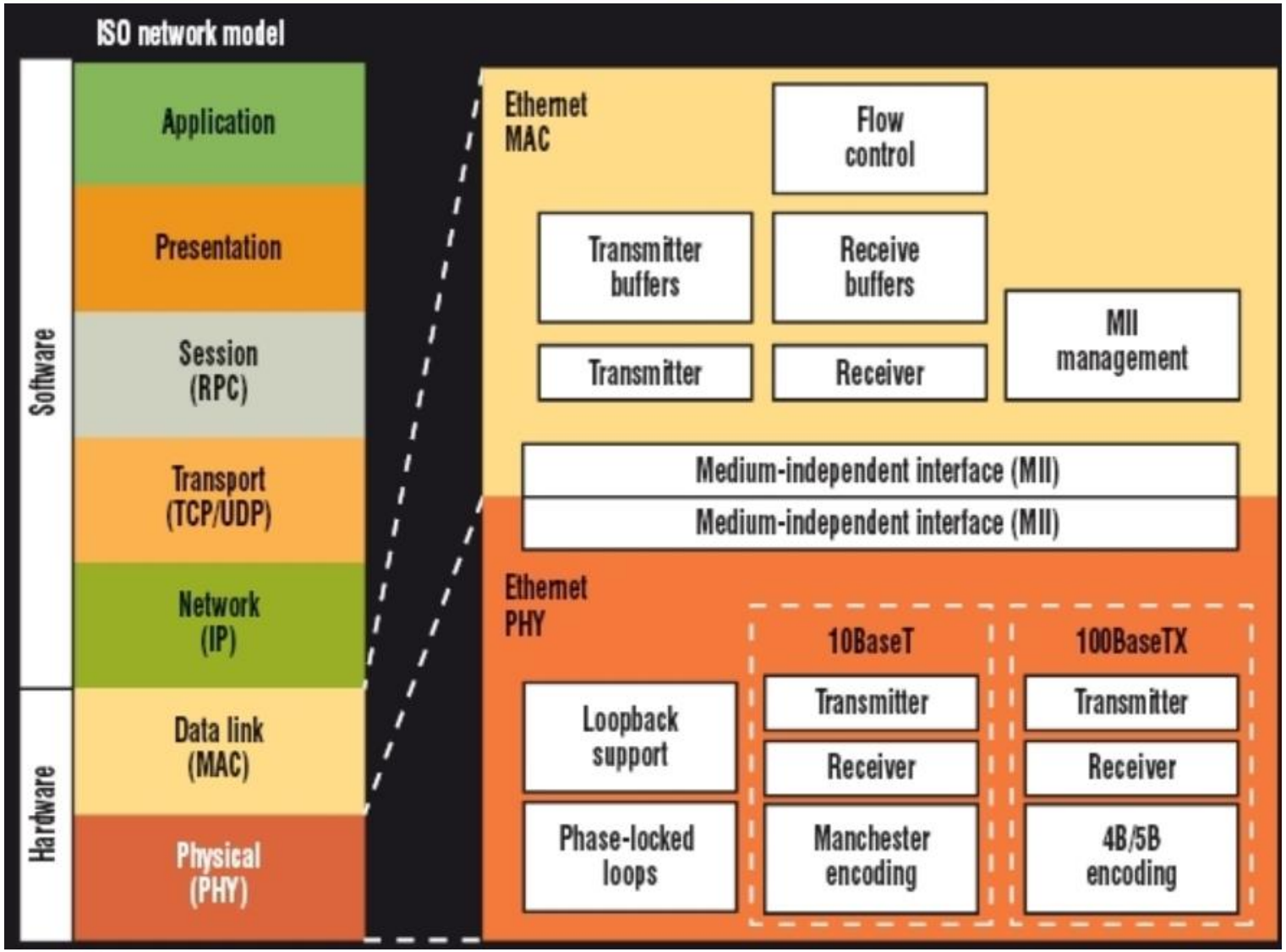
网络协议分层

OSI七层网络模型	Linux TCP/IP 四层概念模型	对应网络协议
应用层	应用层	TFTP, FTP, NFS, WAIS
表示层		Telnet, Rlogin, SNMP, Gopher
会话层		SMTP, DNS
传输层	传输层	TCP, UDP
网络层	网际层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层	网络接口	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层		IEEE 802.1A, IEEE 802.2

网络设备驱动程序设计概要

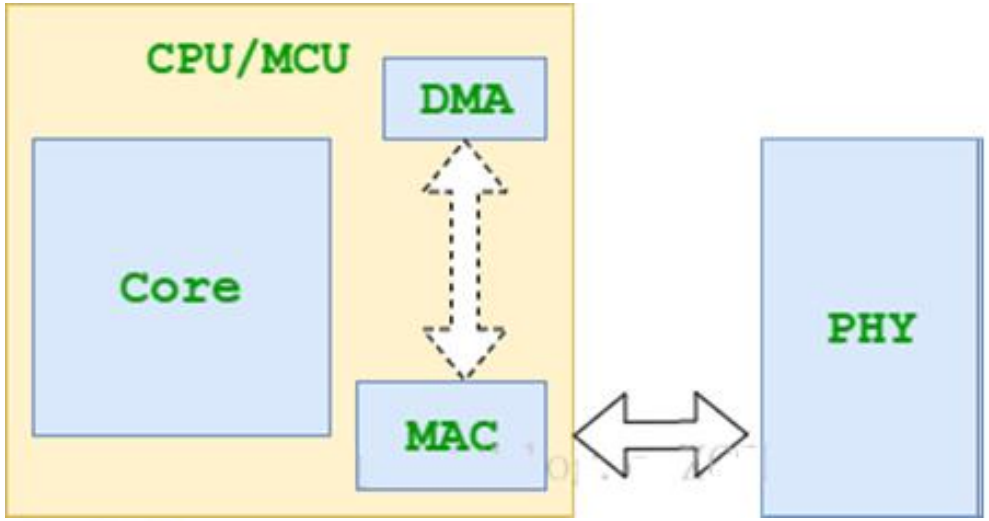
- 以太网络接口对应于ISO网络分层中的数据链路层和物理层
- 数据链路层分为逻辑链路控制子层LLC（Logic Link Control）和介质访问控制子层MAC（Media Access Control）
- 以太网网卡工作在物理层和数据链路层，主要由PHY/MAC芯片、Tx/Rx FIFO、DMA等组成，网线通过变压器接PHY芯片、PHY芯片通过MII接MAC芯片、MAC芯片接PCI总线
- MAC芯片不但要实现MAC子层和LLC子层的功能，还要提供符合规范的PCI界面以实现和主机的数据交换
- PHY芯片主要负责数据收发：CSMA/CD、模数转换、编解码、串并转换等

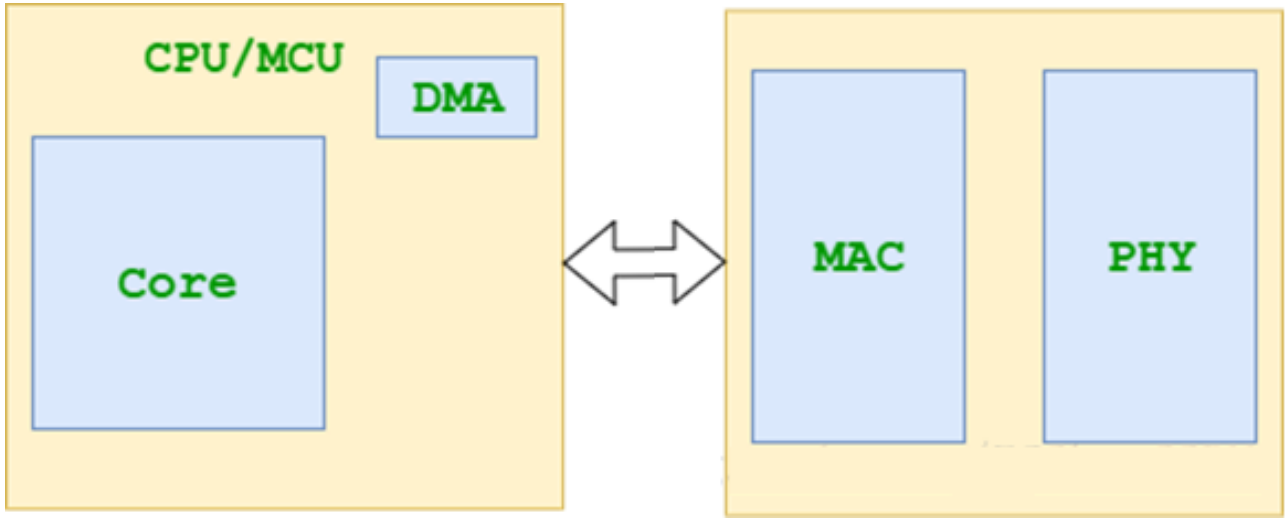
网络设备驱动程序设计概要



网络设备驱动程序设计概要

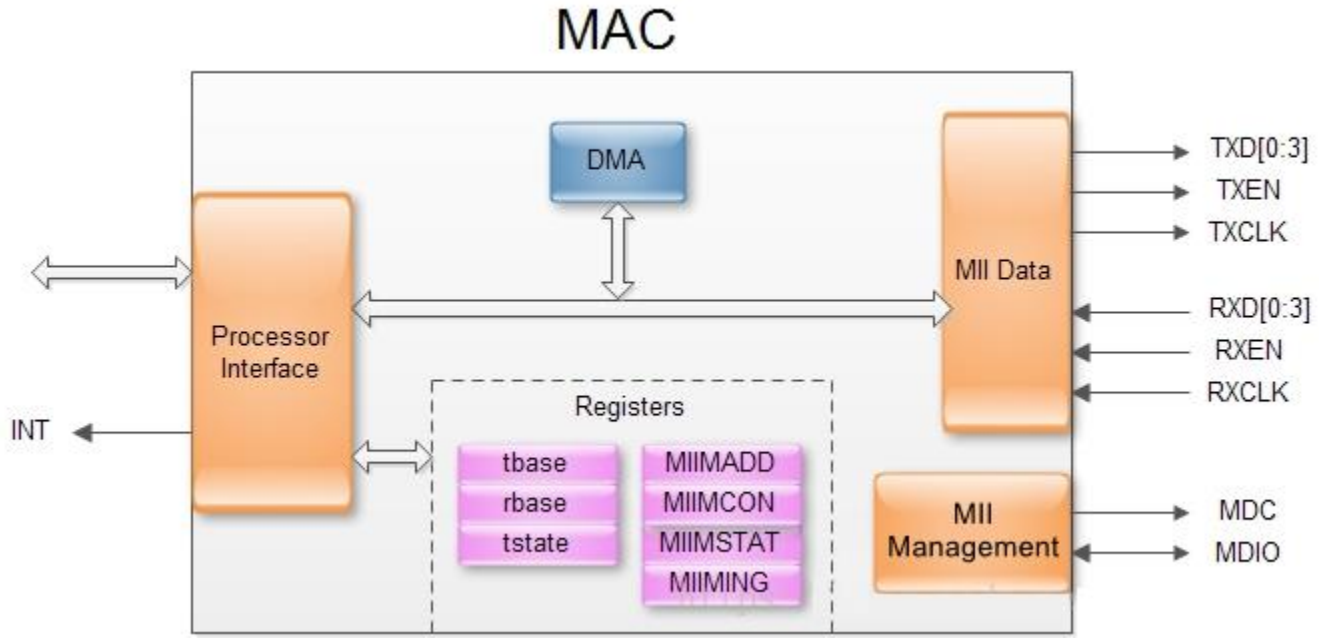
- 在许多嵌入式处理器中都集成了MAC控制器，但是处理器通常是不集成物理层接收器（PHY）的，在这种情况下需要外接PHY芯片，如RTL8201BL、VT6103等





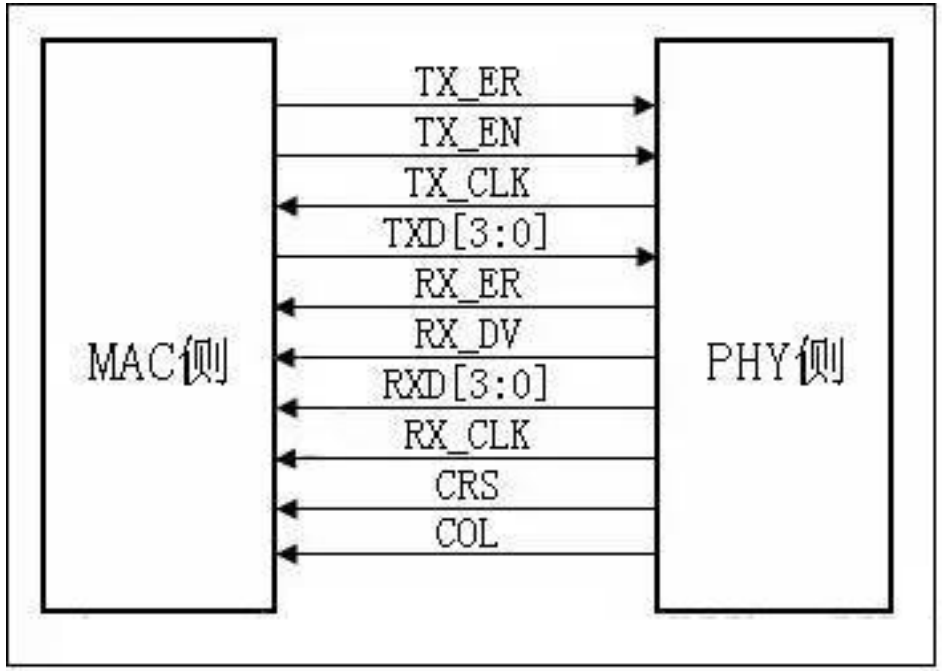
网络设备驱动程序设计概要

- 以太网MAC由IEEE-802.3以太网标准定义
  - 在发送数据的时候，先判断是否可以发送数据，最终将数据以及控制信息以规定的格式发送到物理层
  - 在接收数据的时候，先判断接收的信息并是否发生传输错误，如果没有错误，则去掉控制信息发送至LLC(逻辑链路控制)层



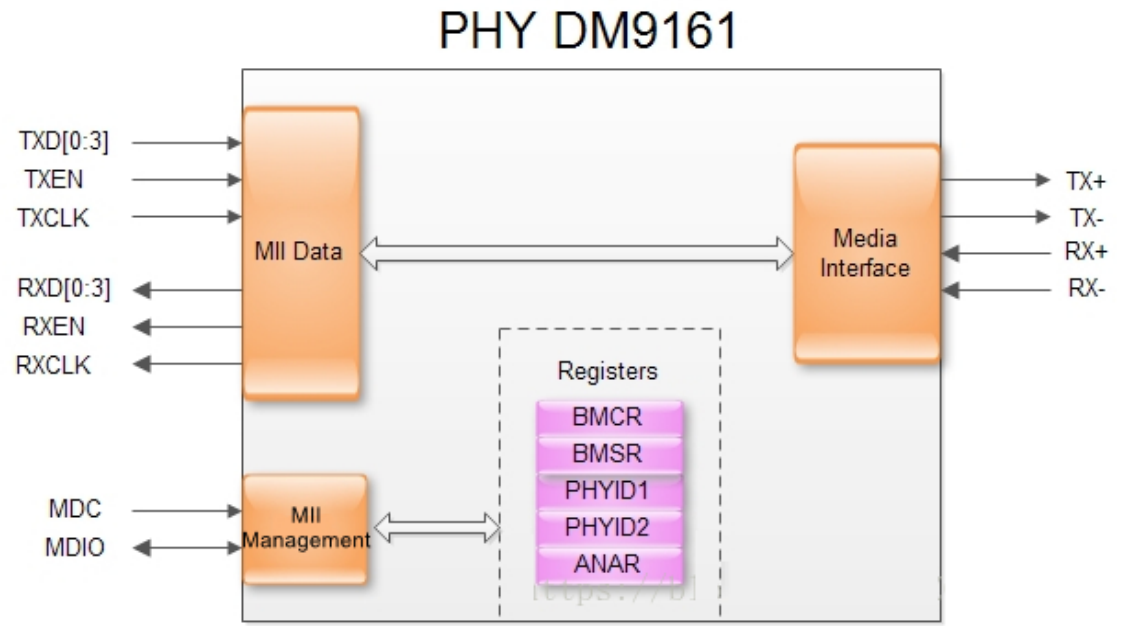
网络设备驱动程序设计概要

- MII是MAC与PHY连接的标准接口，由IEEE-802.3以太网标准定义
  - MII（Media Independent Interface）即媒体独立接口，提供了MAC与PHY之间、PHY与STA(Station Management)之间的互联技术
  - 数据传输的位宽为4位,支持10/100Mb/S。GMII(Gigabit MII)千兆网的MII接口。



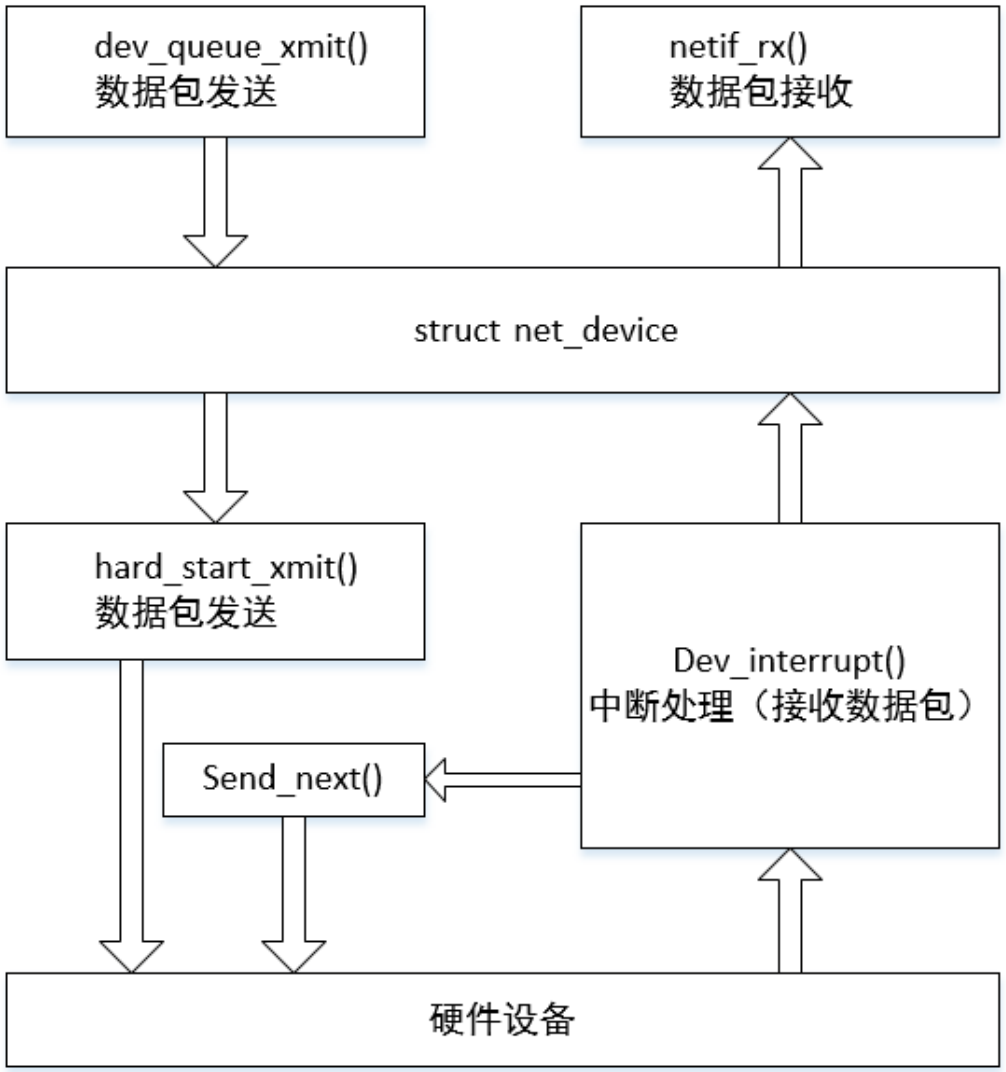
网络设备驱动程序设计概要

- PHY是物理接口收发器，由IEEE-802.3以太网标准定义
  - PHY实现OSI模型的物理层,包括MII/GMII(介质独立接口)子层、PCS(物理编码子层)、PMA(物理介质附加)子层、PMD(物理介质相关)子层、MDI子层
  - STA（station management entity，管理实体,一般为CPU/MCU)通过SMI（Serial Manage Interface）对PHY的行为、状态进行管理和控制



网络设备驱动程序设计概要

网络驱动框架



网络协议接口层

网络设备接口层

驱动程序功能层

网络设备与功能层

网络设备驱动程序设计方法

- 网络设备驱动基本数据结构
- 网络设备初始化
- 打开和关闭接口
- 数据接收与发送
- 查看状态与参数设置

基本数据结构（1）

net_device数据结构

```
char            name[IFNAMSIZ];          /*设备名*/
int             (*init)(struct net_device *dev);     /* 设备的初始化函数，只调用一次 */
unsigned long       mem_end;             /*  内存结束地址 */
unsigned long       mem_start;           /* 内存开始地址*/
unsigned long       base_addr;           /* 内存基地址 */
unsigned int        irq;                 /* 设备中断号 */
unsigned char       if_port;     /* 多端口设备使用的端口类型 */
unsigned char       dma;         /* DMA通道 */
struct net_device_stats    stats;          /* 设备状态信息 */
unsigned            mtu;         /* 最大传输单元，也叫最大数据包 */
unsigned short      type;        /* 接口硬件类型 */
unsigned short      hard_header_len;  /* 硬件帧头长度，一般被赋为ETH_HLEN，即14*/
unsigned char       dev_addr[MAX_ADDR_LEN];          /* 存放设备的MAC地址 */
unsigned char       broadcast[MAX_ADDR_LEN];         /* 存放设备的广播地址 */
```

基本数据结构（2）

操作函数

```
Int    (*open)(struct net_device *dev);
Int    (*stop)(struct net_device *dev);
Int    (*hard_start_xmit) (struct sk_buff *skb, struct net_device *dev);
void (*set_multicast_list)(struct net_device *dev);
int    (*set_mac_address)(struct net_device *dev, void *addr);
int    (*do_ioctl)(struct net_device *dev, struct ifreq *ifr, int cmd);
int     (*set_config)(struct net_device *dev,struct ifmap *map);
int     (*change_mtu)(struct net_device *dev, int new_mtu);
void    (*tx_timeout) (struct net_device *dev);
struct net_device_stats* (*get_stats)(struct net_device *dev);
void    (*poll_controller)(struct net_device *dev);
```

基本数据结构（3）

sk_buffer 数据结构

```
struct sk_buff          *next;      /* 列表中的下一个缓存区 */
struct sk_buff          *prev;      /* 列表中的上一个缓存区 */
struct sock             *sk;        /* 我们所属的socket */
struct net_device       *dev;       /* 我们要到的或者要离开的设备 */
unsigned int            len,        /* 数据包的总长度 */
data_len,    /* 数据包中真实数据的长度 */
mac_len;     /* Mac包头长度 */
__u32                   priority;   /* 包序列优先级 */
__be16          protocol; /*存放上层的协议类型,可以通过eth_type_trans()来获取 */
```

基本数据结构（4）

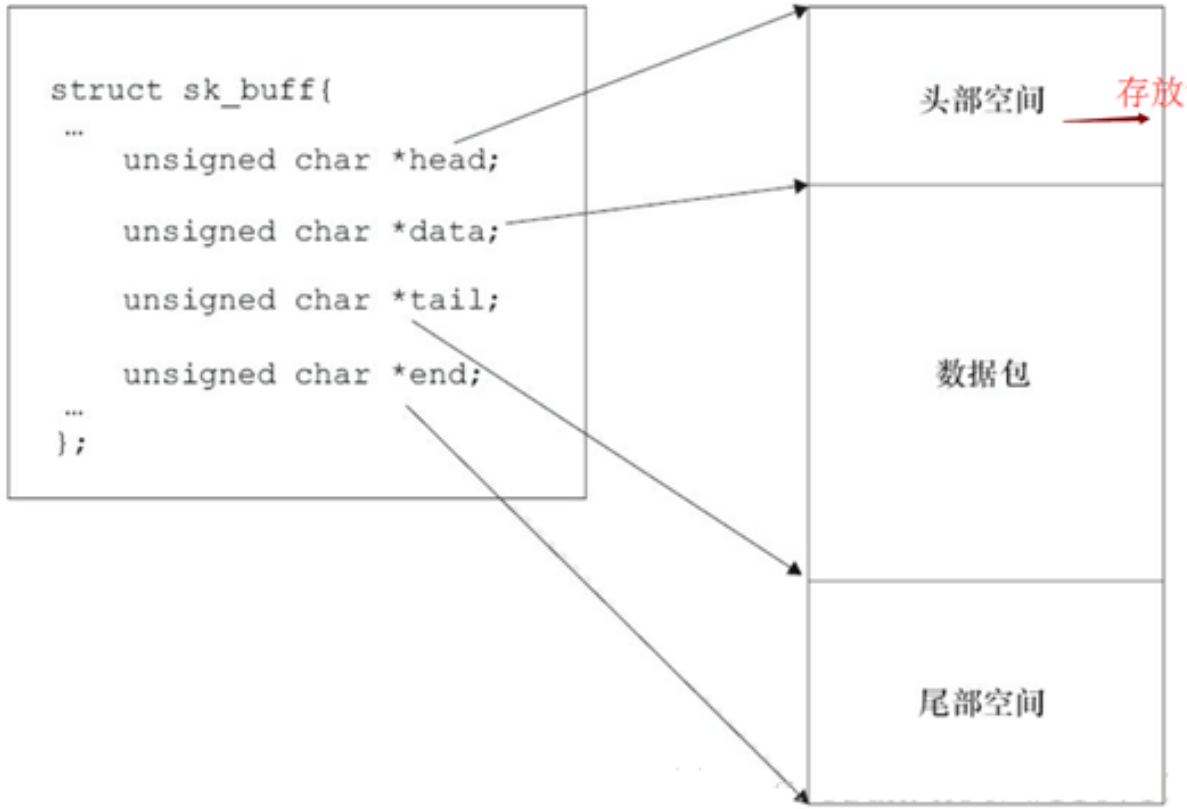
sk_buffer 数据结构

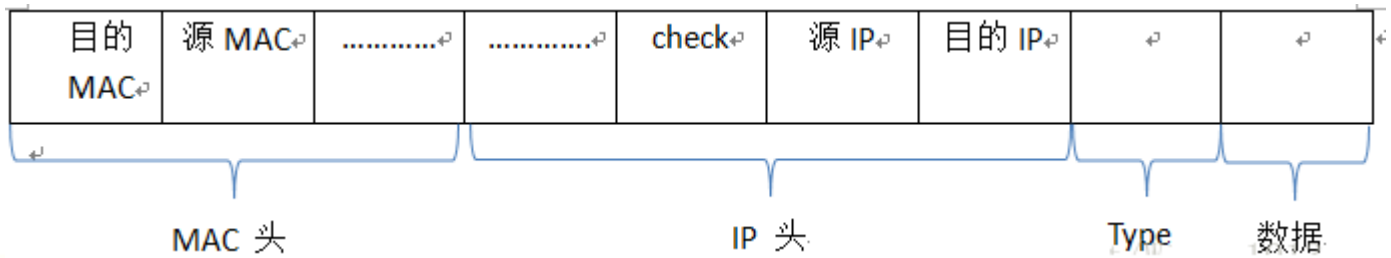
```
sk_buff_data_t          transport_header; /* 传输层头偏移量 */
sk_buff_data_t          network_header;   /* 网络层头偏移量 */
sk_buff_data_t          mac_header;       /* 链路层头偏移量 */

/* These elements must be at the end, see alloc_skb() for details.  */
sk_buff_data_t          tail;     /* 缓存区数据包末尾指针 */
sk_buff_data_t          end;      /* 缓存区末尾指针 */
unsigned char           *head,    /* 缓存区协议头指针 */
                        *data;    /* 缓存区数据包开始位置指针 */
```

基本数据结构（5）

```
struct sk_buff{
 …
    unsigned char *head;
    unsigned char *data;
    unsigned char *tail;
    unsigned char *end;
 …
};
```





基本数据结构（6）

```
/* 分配一个sk_buff结构，供协议栈代码使用 */
struct sk_buff  *alloc_skb(unsigned int size,gfp_t priority);
/* 分配一个sk_buff结构，供驱动代码使用 */
struct sk_buff *dev_alloc_skb(unsigned int len);
/* 释放一个sk_buff结构，供协议栈代码使用。 */
void kfree_skb(struct sk_buff *skb);
/* 将 tail 指针向数据区的末尾移动，增加了 len 字段的长度*/
unsigned char *skb_put(struct sk_buff *skb, unsigned int len);
/*将 data 指针向数据区的前端移动，增加 了len 字段的长度*/
unsigned char *skb_push(struct sk_buff *skb, unsigned int len);
/* 将data 指针向数据区的后端移动，减少了len 字段的长度*/
unsigned char *skb_pull(struct sk_buff *skb, unsigned int len);
/*释放一个sk_buff结构，供驱动代码使用 */
void dev_kfree_skb(struct sk_buff *skb)  /*
```

网络设备初始化

- 主要对net_device结构体进行初始化
- 由net_device的init函数指针指向的函数完成，当加载网络驱动模块时该函数就会被调用
  - 检测网络设备的硬件特征，检查物理设备是否存在。
  - 检测到设备存在，则进行资源配置。
  - 对net_device成员变量进行赋值。

打开和关闭接口

- 打开接口
  - 在数据包放送前，必须打开接口并初始化接口
  - 打开接口的工作由net_device的open函数指针指向的函数完成，该函数负责的工作包括请求系统资源，如申请I/O区域、DMA通道及中断等资源
  - 告知接口开始工作，调用netif_start_queue激活设备发送队列。
- 关闭接口
  - 该操作由net_device的stop函数指针指向的函数完成，该函数需要调用netif_stop_queue停止数据包传送

数据接收与发送（1）

- 数据发送
  - 数据在实际发送的时候会调用net_device结构的hard_start_transmit函数指针指向的函数，该函数会将要发送的数据放入外发队列，并启动数据包发送
- 并发控制
  - 发送函数在指示硬件开始传送数据后就立即返回，但数据在硬件上的传送不一定完成。硬件接口的传送方式是异步的，发送函数又是可重入的，可利用net_device结构中的xmit_lock自旋锁来保护临界区资源
- 传输超时
  - 驱动程序需要处理超时带来的问题，内核会调用net_device的tx_timeout，完成超时需做的工作，并调用netif_wake_queue函数重启设备发送队列

数据接收与发送（2）

- 数据接收
  - 中断方式：当网络设备接收到数据后触发中断，中断处理程序判断中断类型。如果是接收中断，则接收数据，并申请sk_buffer结构和数据缓冲区，根据数据的信息填写sk_buffer结构。然后将接收到的数据复制到缓冲区，最后调用netif_rx函数将skb传递给上层协议，再将报文skb挂到该CPU的softnet_data结构input_pkt_queue队列上。
  - 内核为驱动程序提供一个虚拟设备，叫做积压设备，对应结构softnet_data->backlog_dev。input_pkt_queue即是该设备的积压队列，用于存储skb。
  - 中断上半部只是将报文入队，并将backlog的实例挂到poll_list上，下半部net_rx_action->process_backlog将报文进一步处理

数据接收与发送（3）

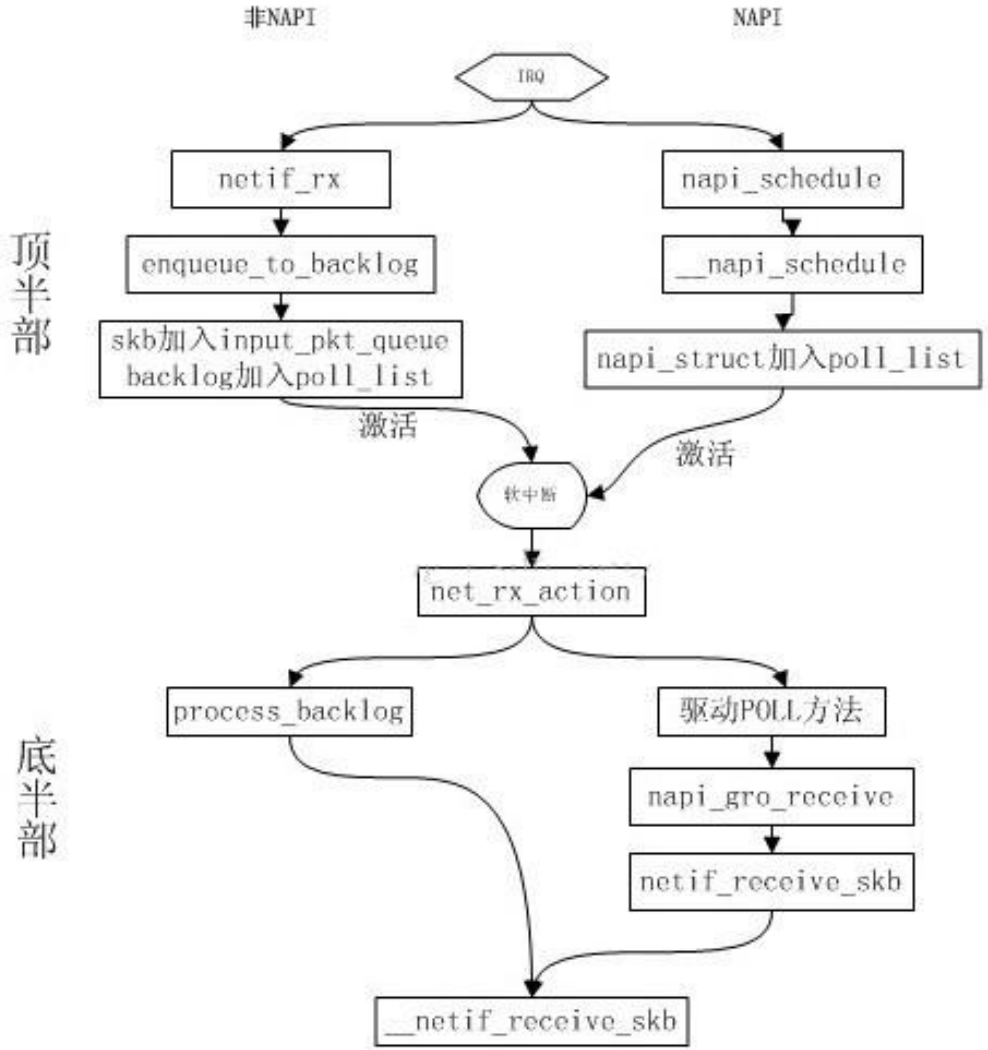
数据接收

- 中断方式的每个报文都触发中断，如果报文太快，中断太频繁，CPU总是处理中断，其他任务无法得到调度，于是NAPI（NewAPI）出现了，采用中断+轮询的方式收包以提高吞吐：在轮询方式下，首个数据包到达产生中断后触发轮询过程，轮询中断处理程序首先关闭“接收中断”，在接收到一定数量的数据包并提交给上层协议后，再开中断等待下次轮询处理。使用netif_receive_skb函数向上层传递数据

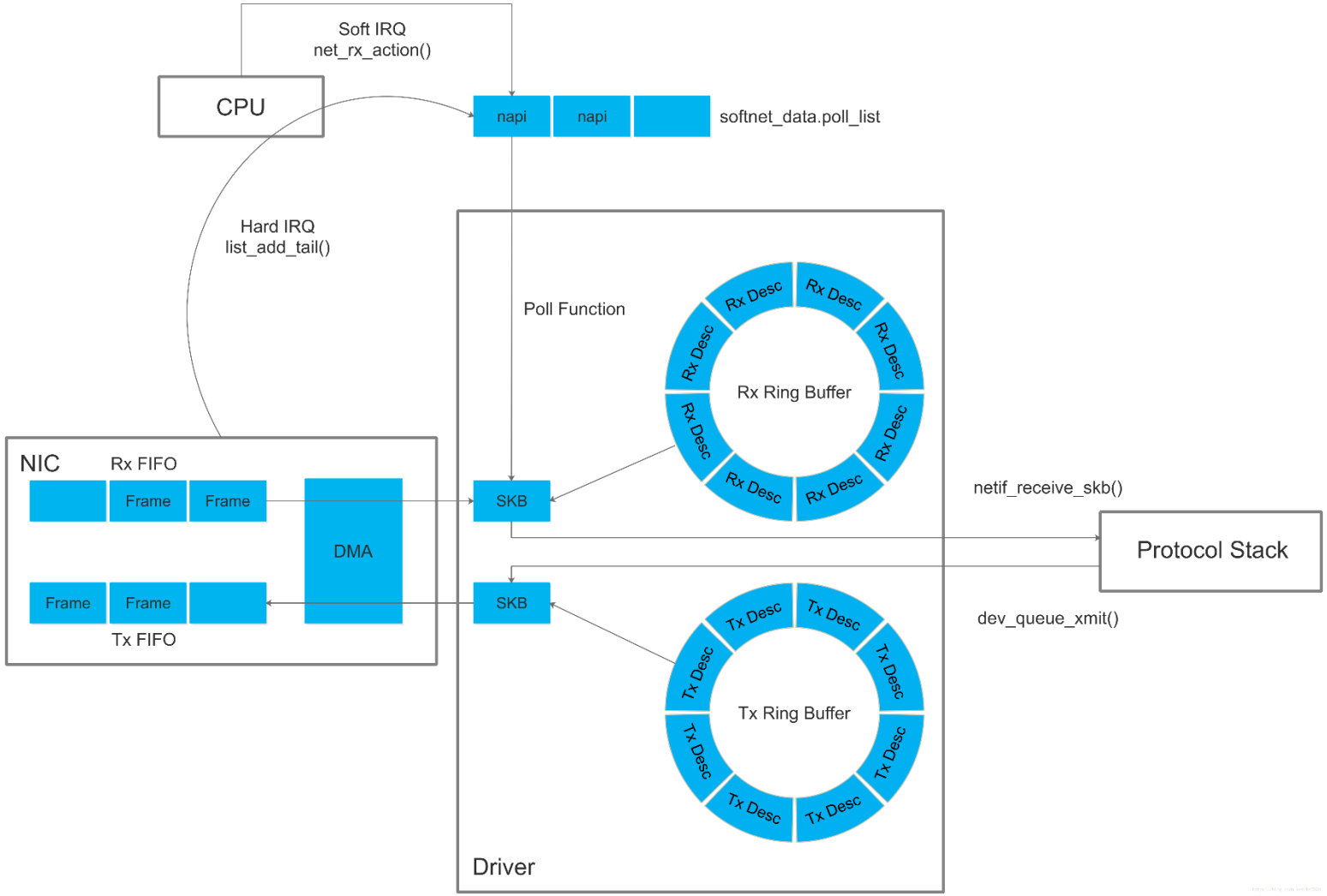
数据接收与发送（4）

- 在收包中断中将网卡napi实例加入softnet_data的poll_list链表上，然后设置NET_RX_SOFTIRQ软中断标志，等待net_rx_action中检查标志并处理。在下面两种情况时，进行轮询收包：
- do_IRQ-->irq_exit-->do_softirq-->call_softirq-->__do_softirq中断上半部退出的时候调用，软中断处理函数net_rx_action，net_rx_action遍历poll_list链表
- __do_softirq循环调用MAX_SOFTIRQ_RESTART（= 10）次net_rx_action如果还有pending的报文，则wakeup_softirqd唤醒ksoftirqd内核线程运行run_ksoftirqd-->__do_softirq-->net_rx_action收包

数据接收与发送（5）


非NAPI
NAPI
IRQ
netif_rx
napi_schedule
顶半部
enqueue_to_backlog
__napi_schedule
skb加入input_pkt_queue
backlog加入poll_list
napi_struct加入poll_list
激活
激活
软中断
net_rx_action
process_backlog
驱动POLL方法
底半部
napi_gro_receive
netif_receive_skb
__netif_receive_skb

数据接收与发送（6）


Soft IRQ
net_rx_action()
CPU
napi
napi
softnet_data.poll_list
Hard IRQ
list_add_tail()
Poll Function
Rx Desc
Rx Ring Buffer
NIC
Rx FIFO
Frame
Frame
DMA
SKB
netif_receive_skb()
Protocol Stack
Frame
Frame
Tx FIFO
SKB
dev_queue_xmit()
Tx Desc
Tx Ring Buffer
Driver

数据接收与发送（7）

- 网络数据发送过程
  - 网卡驱动创建tx descriptor ring
  - 协议栈通过dev_queue_xmit()将sk_buff下送网卡驱动
  - 网卡驱动将sk_buff放入tx descriptor ring
  - DMA通过PCI总线将数据缓存区复制到Tx FIFO
  - 复制完后，通过MAC芯片将数据包发送出去
  - 发送完后，启动硬中断通知CPU释放数据缓存区

数据接收与发送（8）

- 网络数据接收过程
  - 网卡驱动创建rx descriptor ring
  - 网卡驱动为每个descriptor分配sk_buff和数据缓存区，流式DMA映射数据缓存区，将数据缓存区的总线地址保存到descriptor
  - 网卡接收数据包，将数据包写入Rx FIFO
  - 整个数据包写入Rx FIFO后，DMA通过PCI总线将Rx FIFO中的数据包复制到descriptor的数据缓存区
  - 复制完后，网卡启动硬中断通知CPU数据缓存区中已经有新的数据包了，CPU执行硬中断函数__napi_schedule()
  - ksoftirqd执行软中断函数net_rx_action()
  - 网卡驱动通过netif_receive_skb()将sk_buff上送协议栈

查看状态与参数设置（1）

- 链路状态：驱动程序可以通过查看设备的寄存器来获得链路状态信息。当链路状态改变时，驱动程序需要通知内核
  - `void netif_carrier_off(struct net_device *dev);`
  - `void netif_carrier_on(struct net_device *dev);`
- 设备状态：驱动程序的get_stats()函数向用户返回设备的状态和统计信息，保存在一个net_device_stats结构体。
- 设置MAC地址：调用ioctl并且参数为SIOCSIFHWADDR时，就会调用set_mac_address函数指针指向的函数。
- 接口参数设置：调用ioctl并且参数为SIOCSIFMAP时，就会调用set_config函数指针指向的函数，内核会给该函数传递一个ifmap的结构体。该结构体中包含了要设置的I/O地址、中断等信息

查看状态与参数设置（2）

```
struct net_device_stats
{
        unsigned long   rx_packets;     /*收到的数据包数        */
        unsigned long   tx_packets;     /*发送的数据包数        */
        unsigned long   rx_bytes;       /*收到的字节数 */
        unsigned long   tx_bytes;       /*发送的字节数 */
        unsigned long   rx_errors;      /*收到的错误包数        */
        unsigned long   tx_errors;      /*发送的错误包数        */
        unsigned long   rx_dropped;     /* 接收包丢包数*/
        unsigned long   tx_dropped;     /* 发送包丢包数*/
        unsigned long   multicast;      /*收到的广播包数        */
        unsigned long   collisions;
```

查看状态与参数设置（3）

```
/* 详细接收错误信息: */
unsigned long     rx_length_errors; /*接收长度错误 */
unsigned long     rx_over_errors;   /*溢出错误                */
unsigned long     rx_crc_errors;    /*CRC校验错误  */
unsigned long     rx_frame_errors; /*帧对齐错误    */
unsigned long     rx_fifo_errors;    /* 接收fifo错误  */
/* 详细发送错误信息 */
unsigned long     tx_aborted_errors; /*发送中止               */
unsigned long     tx_carrier_errors; /*载波错误               */
unsigned long     tx_fifo_errors;    /*发送fifo错误   */
unsigned long     tx_window_errors;        /*发送窗口错误 */
```

网络设备驱动案例—AT91SAM9G45

- 设备侦测：macb_probe
- 设备打开与关闭：macb_open/macb_close
- 数据的接收：macb_rx_frame
- 数据的发送：macb_tx
- 中断处理：macb_interrupt

□谢　谢！