Linux网络驱动框架

**作者：向仔州**

基于IMX6平台测试，对比2440开发平台

用linux2.6内核思维去开发linux3.14.28内核网卡驱动

网络驱动流程和注册流程……………………………………………………….2

网络包发送………………………………………………………………………………3

网络包接受………………………………………………………………….………….11

IMX6和AR8031 1000M以太网降频100M使用……………………14

网络驱动流程

应用层APP socket()接口

7层网络协议

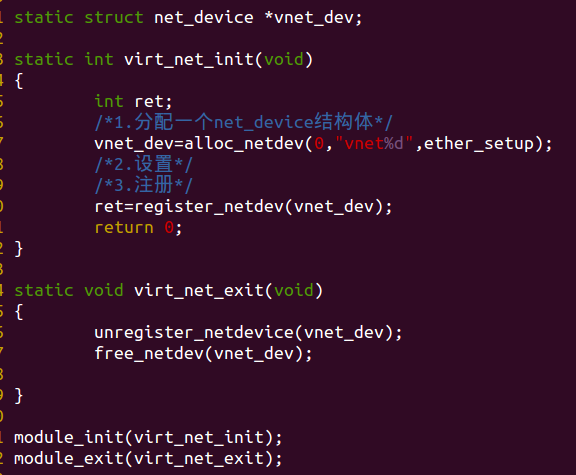
不知道用什么方法就调用到了驱动程序

网线有什么数据过来靠中断函数里面的netif\_rx函数发送给应用层

Socket发送数据包靠Hard\_start\_xmit这个函数发送给硬件

硬件相关的驱动程序

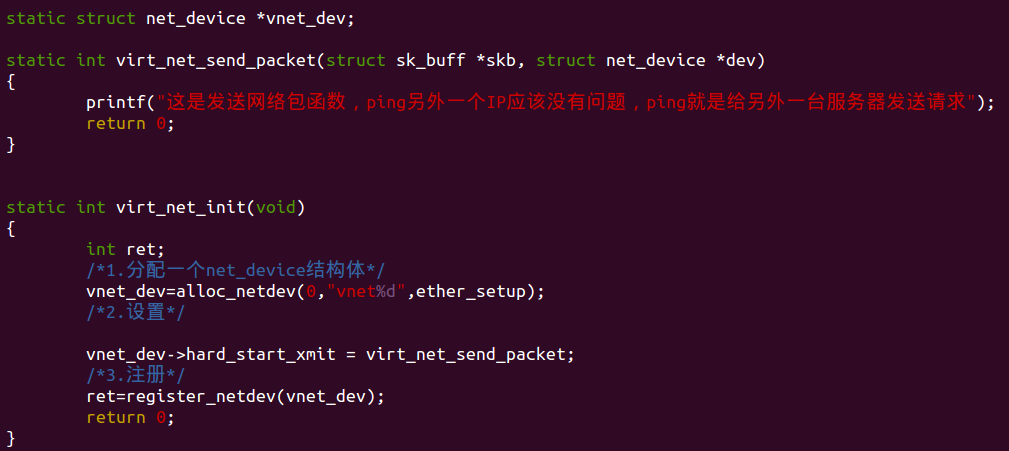
驱动注册流程



这个注册函数在3.14内核上，开发板上电加载这个驱动会报错，所以这里写的代码仅供参考

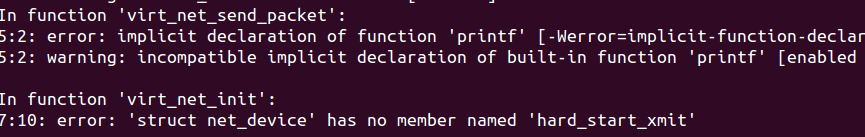
设置设备节点，ifconfig的时候就可以看到这个节点名，这个名字可以自己取

网络包发送



网络包发送函数

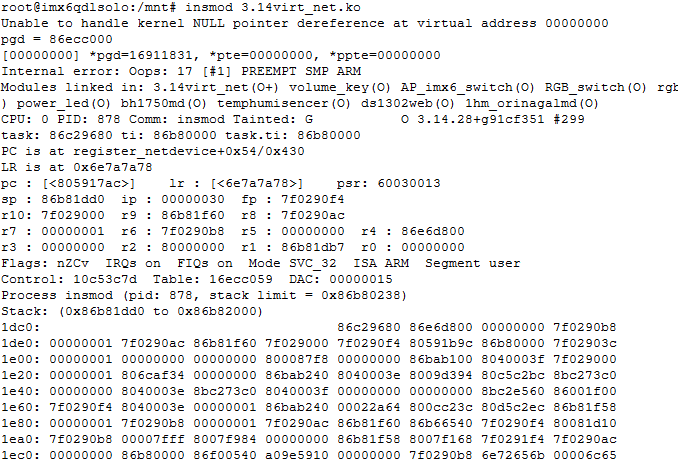
这是2.6内核给发送函数发送网络包的方式



用3.14新内核去编译2.6老内核写的驱动是会报错的，因为net\_device数据结构变了。



这样编译就不会出错

但是insmod内核的时候会崩



那是因为网卡驱动加载内核的时候就会自动去执行发包函数，去请求网线自动分配IP地址，但是我这里没有写这个发包函数所以内核崩了



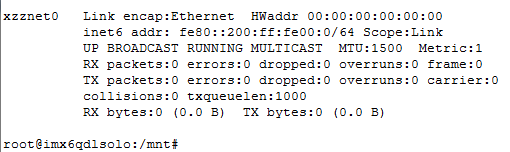
将发包函数加进来



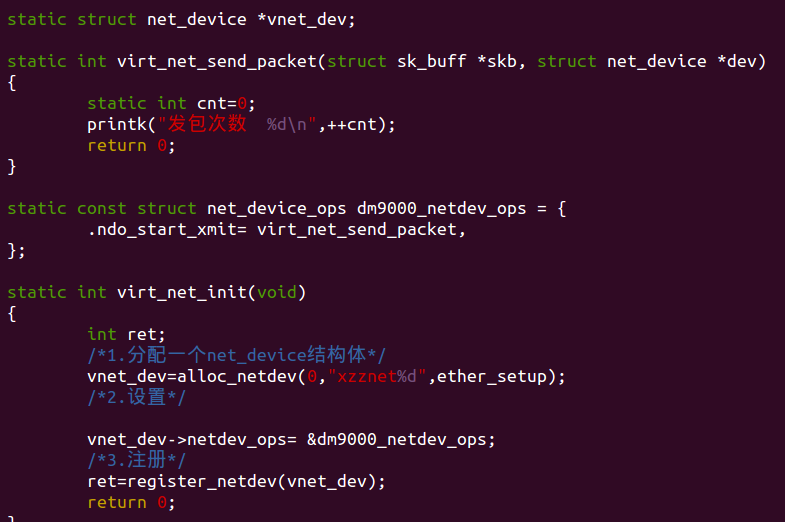
加载虚拟网卡驱动没有问题了



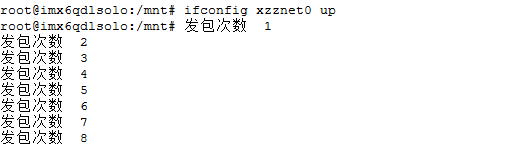
启动我自己设置名字的网卡，驱动程序就会去调用发包函数。



我们现在将发包函数里面的打印改成能正常测试的一些东西



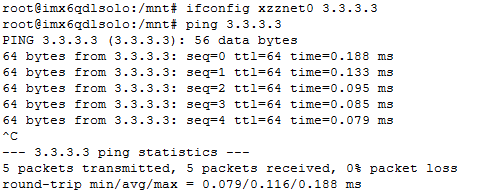
修改发包函数计算发包次数



网卡启动也要发8个包

一包数据8个字节

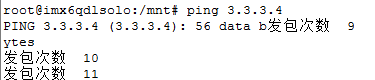
启动自己名字的网卡会出现发包



8 X 8=64

也就是一个包8字节，8个包64位

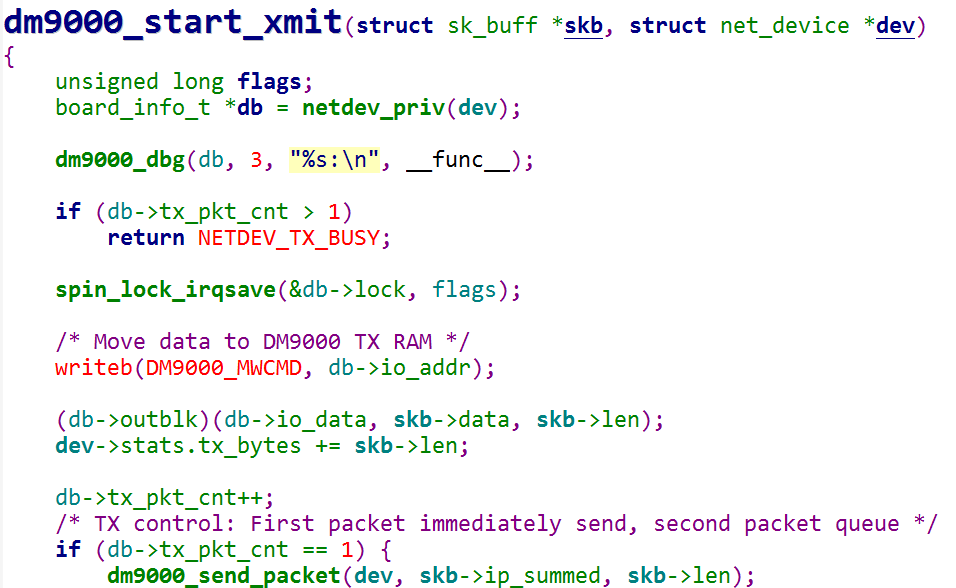
网卡ping自己也是8个包，那么我们ping下别人

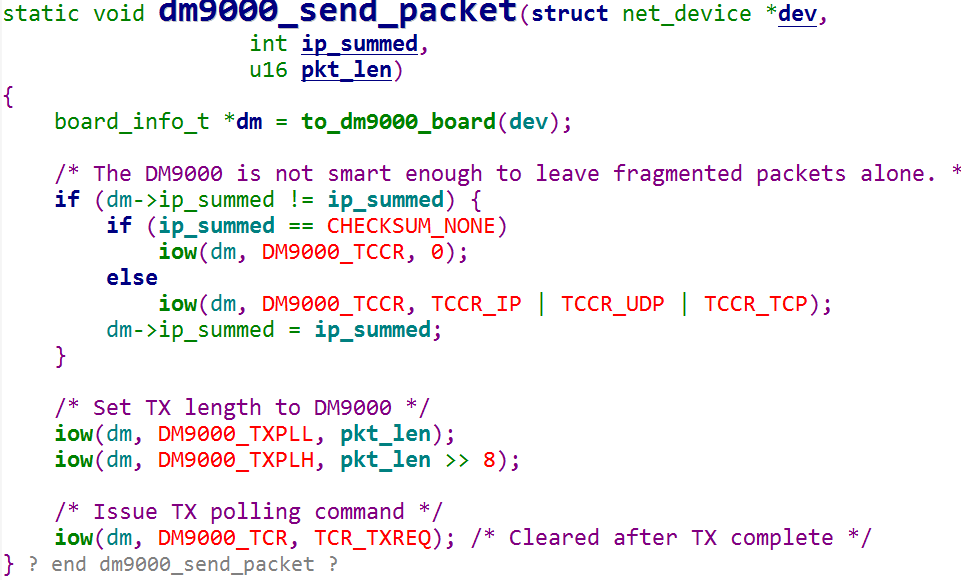
ping别人就是执行发包函数

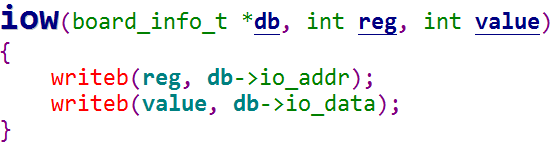
真实网卡发送函数是怎么样的？



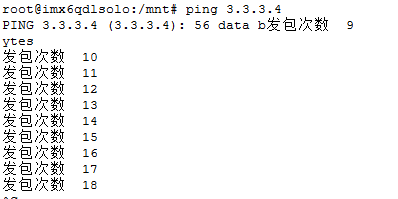
比如像3.14.28内核DM9000网卡这样的做法

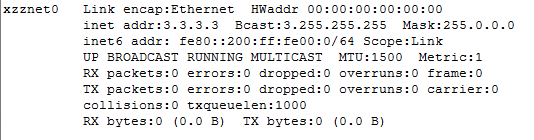




这就是发送给网卡寄存器

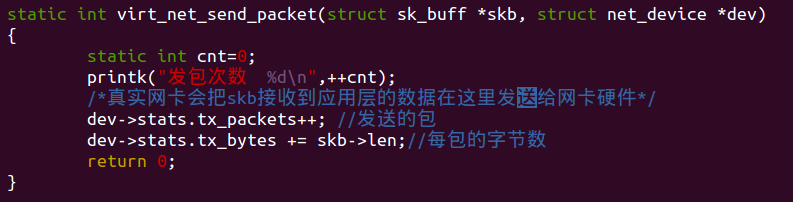
我们现在发现个问题，就是发送数据包之后没有统计信息

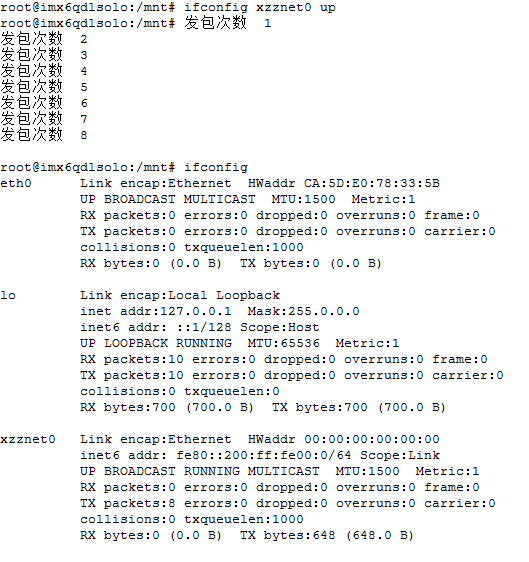




没有发送包数据统计

这个时候我们要修改.ndo\_start\_xmit= virt\_net\_send\_packet,发送函数里面的代码

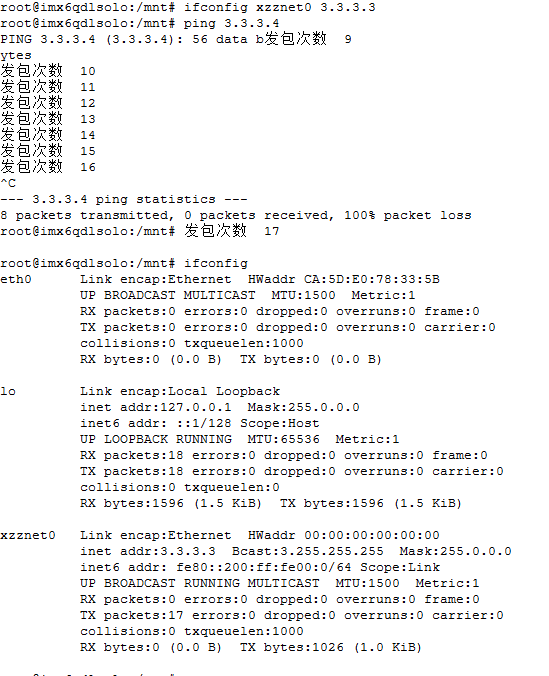




8个包648个字节

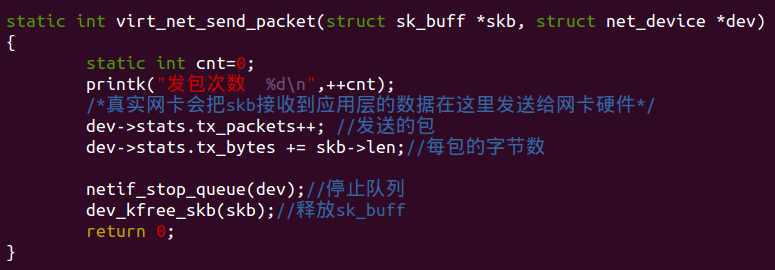
统计为8个包

网卡启动发了8个包

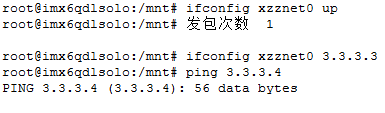


我ping外网有接着发包，现在发了17个包，证明数据发送包统计功能搞定。

每次发完一个包之后我们网卡都要停止队列，释放skb\_buff，为下一次发送留空间



增加这两段代码



但是停止队列之后，发送函数只执行一次然后就再也无法重复执行了，上面那种启动网卡发送8个字节才是正常的啊，这里为什么出现这种情况？

那是因为你发送一包数据后停止队列，但是你没有再次唤醒队列，所以发送函数无法执行

网卡芯片

CPU用.ndo\_start\_xmit= virt\_net\_send\_packet函数发应用层的一包数据给网卡芯片

发送一包数据

网卡芯片

出问题了

CPU马上跟着再发一包数据给网卡芯片

发送一包数据

为什么第二包数据网卡芯片出问题呢？因为你CPU根本不知道网卡芯片接受CPU上一包发来的数据处理完没有，又鼓捣给别个网卡再发一包，你说网卡芯片涨不涨得死。

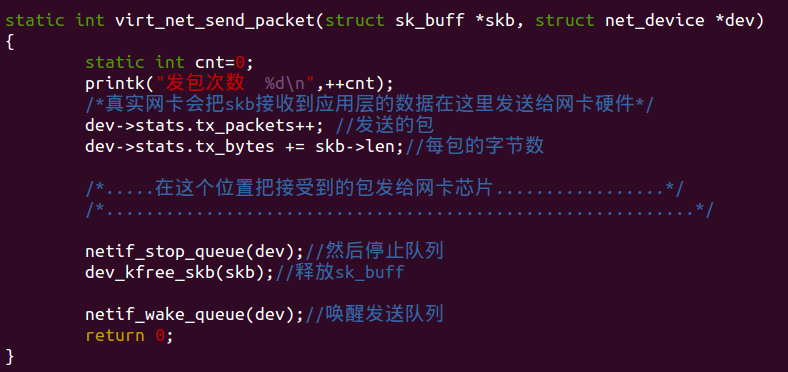
所以停止队列让CPU稍等一下

网卡芯片

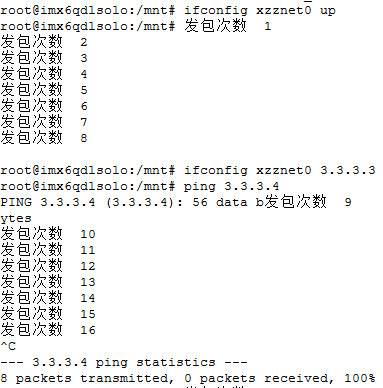
处理完上一包数据之后，发送中断给CPU

CPU停止了发送队列等到网卡芯片的中断

中断信号



这里是为了方便虚拟网卡实验，实际这个唤醒队列代码要放到网卡发给CPU的中断函数里面

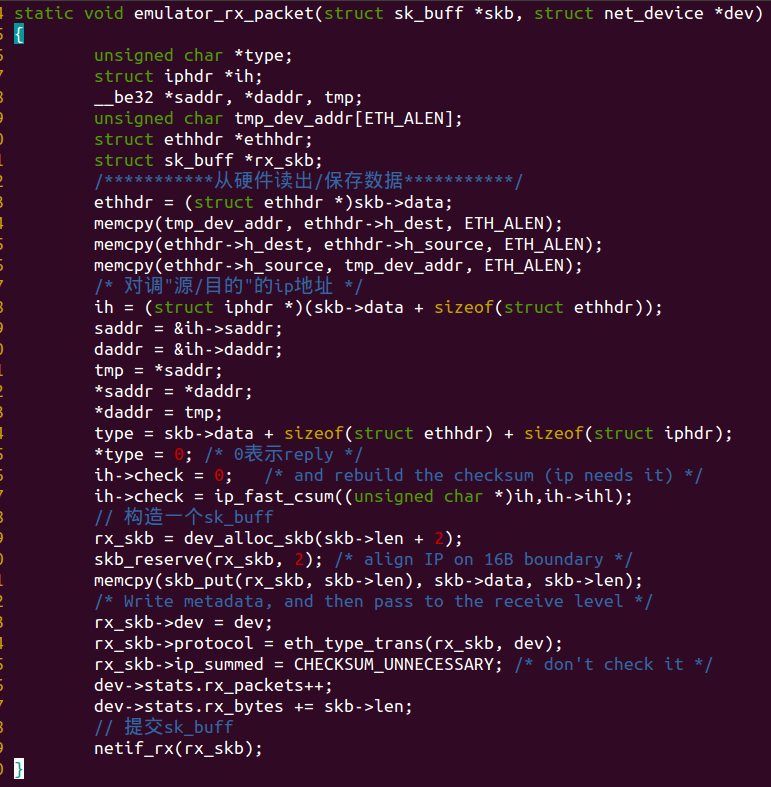


这里记录每次ping的发包数量，而不是总共发了多少包的数量

网络包接受



我们假装把发出去的数据包skb向接受包里面扔



将rx\_skb发给应用层，其实rx\_skb就是发送出去的skb

接受程序提取skb给rx\_skb

其实这个发包收包模拟过程是这样的

应用层发送命令

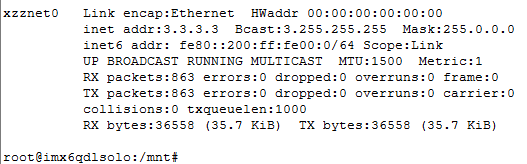
又回传给应用层

Xmit函数里面写入netif\_rx函数

这里和外网是断开的

这里和外网是断开的

网卡芯片

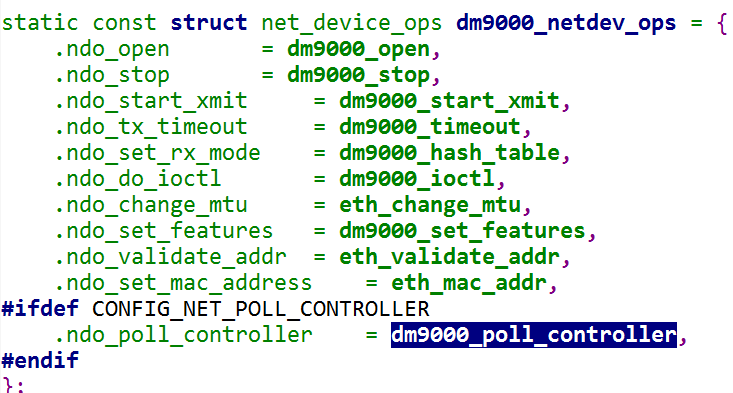


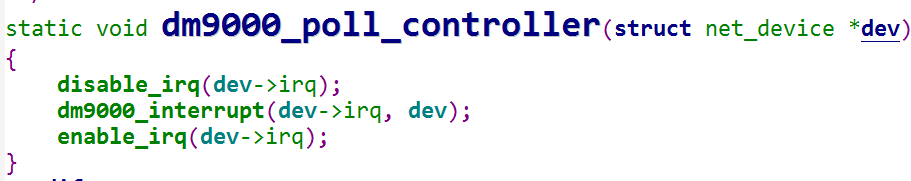
这样只是为了能让RX接受字节数量也能统计出来，其实没有任何意义。

实际的网卡驱动里面是不能在xmit函数里面写netif\_rx函数的

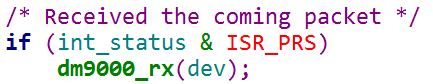
所以我们这里实现的网络接受就是将发出去的数据又接受回来，模拟老一个过程。实际过程中netif\_rx()函数绝对是放在芯片接受中断类函数里面的。

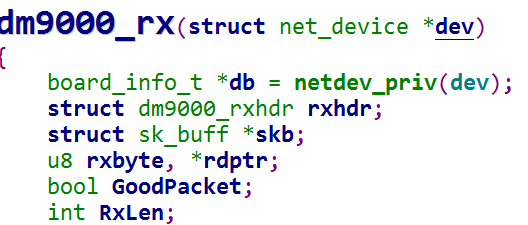
比如3.14.28内核的DM9000网卡驱动

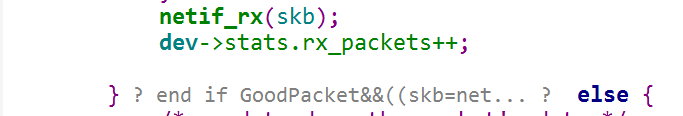










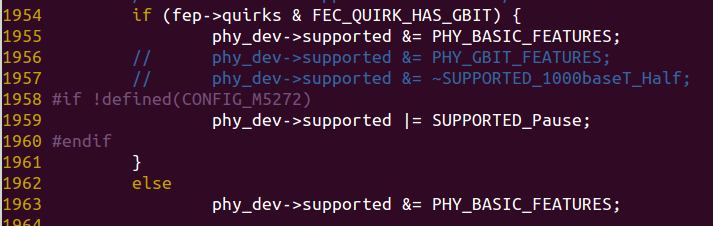


IMX6以太网AR8031降频处理

因为硬件问题，导致AR8031网口端也就是水晶头端无法实现1000M传输，但是100M传输是很正常的，所以为了让开发板插入1000M路由器，也能识别成100M，需要修改内核代码。

在这个路径下

打开这个C文件



将1956和1957行代码屏蔽掉，把1963行的代码复制上去。这样就实现了进入1000M也初始化100M的代码，进入100M也是初始化100M的代码。

然后编译内核，烧写。