GMAC 设计指南

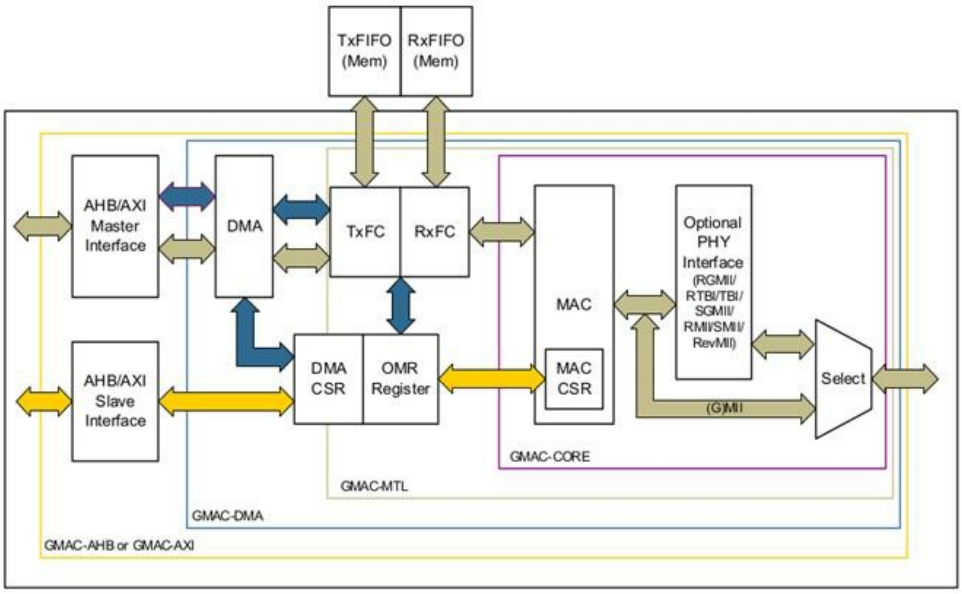
综述

目的

本篇文档重点描述基于SF16A18 芯片开发的GMAC驱动，所以使用kernel版本是 3.18.29，驱动架构已经将系统相关的部分进行了抽象处理，所以可以轻松的进行移植。

硬件简介

下图为gmac硬件架构介绍



- GMAC-AHB
  GMAC-AHB 模块通过AHB master接口向系统的内存传输数据。系统的cpu通过64位宽的AHB slave接口来访问GMAC 内部的控制
  寄存器和状态寄存器
- GMAC-DMA
  DMA模块有独立的收发功能，还有独立的寄存器，发送功能将数据从系统内存搬运到设备端口（MTL）,接收模块将数据从设备端口搬运到
  系统内存中，控制器通过描述符来高效的进行数据收发，尽可能的减小CPU的介入。DMA被设计为一个面向数据包的数据收发模块，也就是以太网传输
  中的一帧，控制器可以以设置为每次一帧收发完成后向cpu发送中断，或者其他正常或异常情况下触发中断。
- GMAC-MT
  MTL模块由两组FIFO组成，发送FIFO可以软件控制阈值，接口FIFO也可以使用软件控制阈值（默认64字节）。

- GMAC-CORE
  MAC根据PHY芯片不同支持多种接口，PHY接口智能在reset之后设置一次，MAC应用层通信基于MAC发送接口（MTI),MAC接受接口（MRI）和MAC控制接口
  （MCI）。

MAC 功能

GMAC的功能被划分成下面几个类别
1）协议标准功能
2）MAC
3）DMA模块
4）收发层(MTL)

- 协议标准功能

```
IEEE 802.3-2008 for Ethernet MAC

IEEE 1588-2008 standard for precision networked clock synchronization

AMBA 2.0 for AHB master and slave ports

RGMII/RTBI specification version 2.6 from HP/Marvell

RMII specification version 1.2 from RMII consortium
```

- MAC

```
MAC层支持下列功能
10、100、100Mbps 传输速度，通过下列phy接口
    RGMII 接口，  和外千兆 PHY进行通讯
    RMII接口，和外部高速以太网PHY进行通信

全双工模式
    支持IEEE802.3x flow control 在对输入进行流控的是否，自动发送zero-quanta Pause fra
    可以选择是否将接收的Pause frames转发给用户程序

半双工模式
    支持CSMA/CD Protocol
    Flow control using backpressure support (基于 implementation-specific white p
    半双工千兆模式下帧突发和帧拓展

发送中前导和帧数据起点（SFD）插入

接受中前导和帧数据起点（SFD）删除

基于每一帧进行自动CRC校验和数据填充控制

接收帧进行填充数据和CRC数据剥离

灵活的地址过滤模式：
    31个额外的48 bit 目的地址（DA）过滤器，每字节都可以设置掩码。
    31个48 bit 来源地址（SA）对比检查，每字节都可以设置掩码。
```

```
        针对广播或者单播（DA）地址的256 bit的hash过滤器
        可以选择通过所有多播帧
        用于网络监听的混杂模式接收所有帧，不进行任何filter
        传入所有报文并含有状态报告

    支持可编程帧长度，支持标准或者jumbo 以太网帧，最大长度16KB

    可以编程帧间隙（IFG）（40-96 bit 时间间隔 最单元8 ）

    可选择使用发送帧的时候减小前导帧长度

    对于发送和接收报文有独立的32bit 状态

    识别接受报文的IEEE 802.1Q vlan tag

    额外报文过滤：
        根据VLAN TAG ：  完全匹配和基于HASH过滤
        基于层三和层四进行过滤，  IPV4 IPV6上的TCP和UDP 4个L3 1个L4

    网络状态统计（可选）RMON/MIB计数（RFC2189/RFC2665)

    可以选择侦测远程唤醒帧和AMD幻数包

    可以选择硬件checksum 对于接收的封装在以太网帧中的IPV4和TCP进行校验（TYPE 1）

    可以选择强化接收模块。对于ipv4 头部的checksum 以及封装在IPv4?IPv6中的TCP和UDP iCMP 报文

    可以选择支持以太网帧的时间戳功能，遵照协议IEEE 1588-2002 和 IEEE 1588-2008,对于发送或者

    MDIO master接口来对接PHY设备，进行配置和管理

    发送对于每一帧可以进行CRC 替换 ，原地址插入或者替换，VLAN的插入替换删除等功能

    可以选择4个通用主题的输入或者输出
        通用主题的输入可以编程为在上升沿还是下降沿触发中断
```

- DMA BlocK
  DMA模块在系统内存和MTL之间传输数据，host可以使用DMA寄存器（DMA CSR）来控制DMA操作。
  DMA模块包含一下功能：

64bit数据传输

单channel的发送和接收模块

全同步设计，工作在一个系统时钟下（除了给CSR单独配一个CSR时钟源）

对于面向报文的DMA传输同时带有帧分隔符进行了优化

支持按照字节对其的数据buffe地址

支持双buffer （ring）或者 链表（chained） 的描述符链。

描述符的结构设可以一次传输大量数据，而不需要cpu太多参与（每个描述符最多可以传输8K数据）

针对正常操作或者收发错误有全面的状态报告

可以独立配置的burst大小，可以让DMA模块达到最优化使用host总线。

可以编程的各种情况下的对于中断的选择

针对每帧的收发可以完全进行中断控制

在收发模块间使用Round-robin 或者 fixed-priority arbitration算法调度

对于寄存器访问和数据接口分开为单独接口

- 收发层（MTL）
  MTK模块由两组fifo组成，带有可编辑阈值的发送fifo，和带有可编辑阈值的接收fifo（默认64 byte）。
  MTL层支持下面功能：

64位的收发模块（将应用层和GMAC-CORE连接在一起）

128字节，256字节，512字节，1KB，2KB，4KB，8KB，16KB或者32KB接收端FIFO大小

在GMAC-MTL配置中可选择接口在MTL Rx FIFO的顶端标示接受侦的长度。

在GMAC-MTL配置中可编程burst长度，用于启动burst，最大为MTL Rx和Tx FIFO的一半

EOF传输后将接收状态向量插入接收FIFO，这使得接收FIFO中的多帧存储无需另一个FIFO来存储这些帧的

Cut-Through或Threshold模式下的可配置接收FIFO阈值（默认固定为64字节）

可以选择在接收时过滤所有错误帧，而不是在存储转发模式下将它们转发到应用程序

可选择转发尺寸不足的正确帧

通过为接收FIFO中丢弃或损坏（由于溢出）的帧生成脉冲来支持状态统计

传输FIFO大小可以为256字节或512字节或1KB，2KB，4KB，8KB或16KB

存储和转发机制用于传输到MAC

发送缓冲区管理的可以设置阈值

随时可以配置在FIFO中存储的帧数，默认值是两个帧（固定）包含内部有DMA，GMAC-MTL配置最多有八个

根据接收FIFO填充情况（设定阈值）来自动生成暂停帧或反压信号到MAC

发送中自动重发碰撞帧

丢弃晚期碰撞，过度碰撞，过度延期和欠载情况下的帧

软件控制清除TX FIFO

非活动状态下禁用数据FIFO RAM芯片选择非活动状态以降低功耗

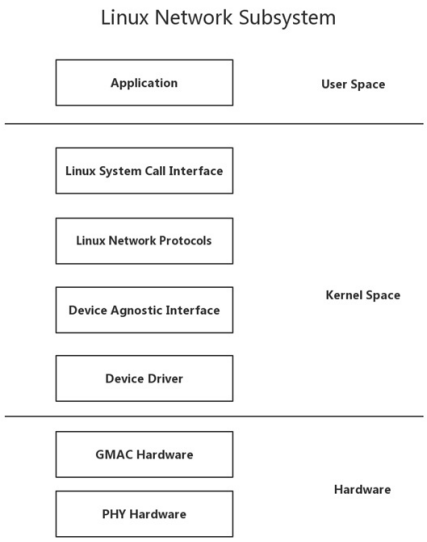
可选模块用于计算并插入以存储转发模式传输的帧中的IPv4报头校验和和TCP，UDP或ICMP校验和

Linux网络堆栈的高级视图

Linux将内核空间的网络级别划分为4个子层。 Linux网络子系统的顶部是Linux系统调用接口层。它为用户提供了访问内核网络子系统的套接字。下面是Linux网络协议层，其中包括TCP，UDP，IP和其他协议。还有一个设备不可知界面层，它提供了协议和设备驱动程序之间的通用接口。最后是控制MAC和PHY硬件的设备驱动层。

Device Agnostic Layer将协议连接到不同的网络驱动程序，并为底层网络设备驱动程序提供一些可以操作高级协议栈的常用功能。如果网络协议层需要将数据包传输到设备驱动程序，则需要调用dev_queue_xmit()函数，该函数对数据包队列进行排序，并从设备驱动程序层调用ndo_start_xmit()函数来完成传输。接收通常由napi_schedule()函数运行。当底层设备收到中断时，它会调用

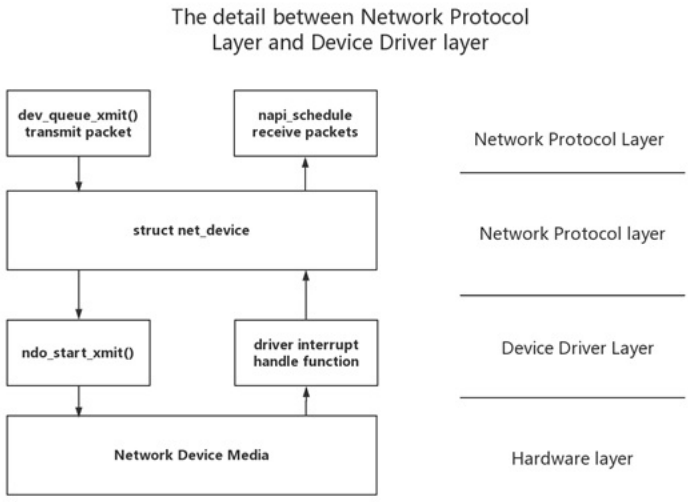
napi_schedule9()函数将数据传输到设备不可知界面层。



网络协议层和设备与媒体层之间的细节

网络接口层为网络协议提供统一的分组收发器接口。 无论协议是ARP还是IP，它总是通过dev_queue_xmit()函数发送数据，并通过napi_schedule()函数接收数据。 该层使网络协议独立于特定设备。

网络设备接口层提供了一个统一的结构net_device，用于向协议接口层描述特定的网络设备特性和操作，协议接口层是设备驱动程序功能层中每个功能的容器。



设备驱动程序层包含结构net_device的具体成员，并且很难完成适当的操作。 它通过ndo_start_xmi()函数发送数据，并通过中断句柄函数接收数据。

网络设备和媒体层是完成数据包收发的物理实体，包括网络适配器和特定的传输介质。 它们由设备驱动程序层中的功能驱动。

Siflower GMAC 驱动

驱动架构

GMAC设备驱动程序包含3个文件：sgmac.c，sgmac.h，Makefile。

在sgmac.c中，有描述符操作，平台功能，设备功能和网络层相关接口，而在Makefile中，有编译的方式（与NPU在同一个Makefile中）。

我们把所有这些东西放在一个文件中，因为没有太多的东西，但它仍然是一个不好的行为。 也许我们会进一步重建架构。

设备驱动程序层包含结构net_device的具体成员

设备驱动程序是使用前面部分提到的驱动程序代码的平台和操作系统设备不可知层开发的。

Linux首先将GMAC设备视为平台设备。 这由platform_driver结构表示。该设备接口的重要成员在下面给出。

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        ...
};
```

Linux中的网络堆栈将GMAC设备视为网络接口/设备。 这由net_device结构表示。 下面给出了此设备接口的重要成员。

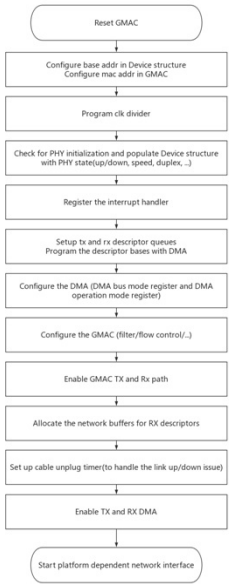
```
struct net_device {
int (* ndo_open) (struct net_device *dev);
int (*ndo_stop) (struct net_device *dev);
int (* ndo_start_xmit) (struct sk_buff *skb, struct net_device *dev);
...
}
```

Siflower GMAC Device Driver将其功能注册到Linux，如下所示。

```
static const struct net_device_ops sgmac_netdev_ops = {
.ndo_open = sgmac_open,
.ndo_stop = sgmac_stop,
.ndo_start_xmit = sgmac_xmit,
...
}
```

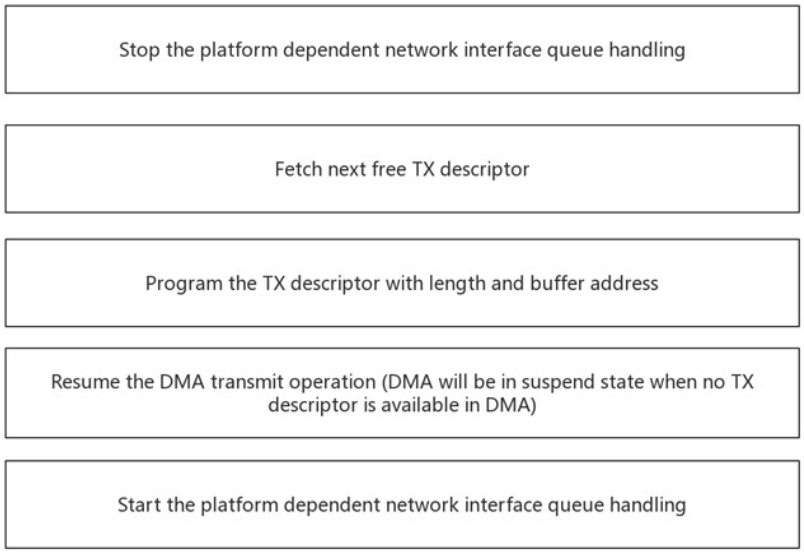
GMAC的初始化顺序

本节给出GMAC驱动程序初始化的简要流程图。 这里提供的步骤作为GMAC初始化的流程。



TX通路的编程指南

537/5000
发送路径在OS准备好发送缓冲区时开始。 内核调用/调用在netdev-> ndo_start_xmit处注册的函数。 在驱动程序注册函数（sgmac_xmit）中执行的操作以流程图的形式提供。 传输在正常执行上下文中处理，这只不过是在进程上下文中，以及在中断上下文中部分地传送（将传送的缓冲存储器交给OS）。 详细描述请参考中断服务程序序列。



中断服务程序编程指南

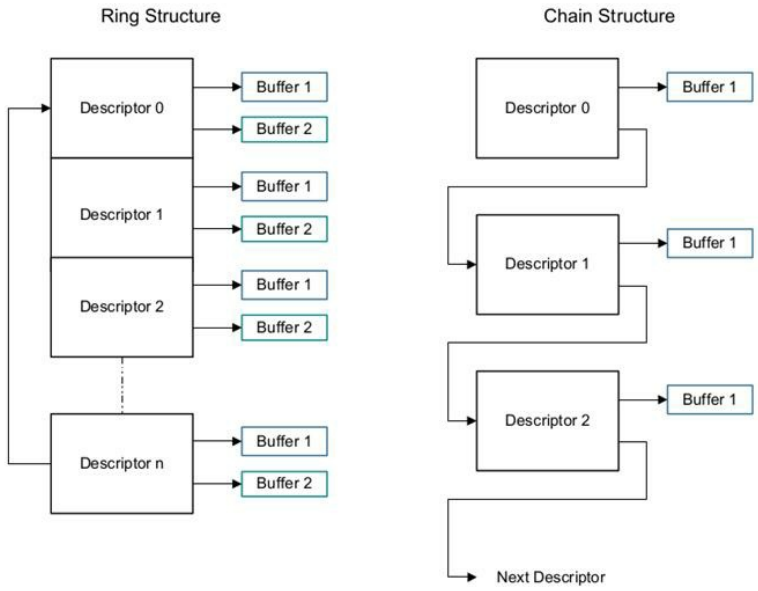
在Linux中，接收是通过中断进行的，而不是在内核中注册处理收到的数据包的函数。 接收完全在内核注册的中断服务例程中处理。 设备生成传输完成中断（在传输数据包后生成，详情请参阅传输路径），表示数据包传输完成。

注：流程图仅解释在发送和接收过程中没有错误的情况下的操作顺序。 出现错误时，应妥善处理，以免发生系统故障。

描述符

295/5000
GMAC支持描述符的环和链结构，而在驱动程序中，我们选择描述符的环模式。 我们提供了Both发送器和Received描述符队列，每个队列都包含64个描述符。 要更改描述符的数量，请相应地更改sgmac.c文件以进行以下操作。

```
#define DMA_TX_RING_SZ   64
#define DMA_RX_RING_SZ   64
```



注意：所提供的中断服务程序无法处理指向环形操作模式下的数据缓冲区的buffer2，因此我们不会在设备驱动程序中使用它。

主机数据缓冲区对齐

TX和RX数据缓冲区对起始地址对齐没有任何限制，缓冲区的起始地址可以与四个字节中的任何一个对齐。 但是，DMA总是启动写传输，地址与总线宽度对齐，并且字节通道中的虚拟数据（旧数据）

无效。 这通常发生在传输以太网帧的开始或结束时。 设备驱动程序应根据缓冲区的起始地址和帧的大小丢弃虚拟字节。

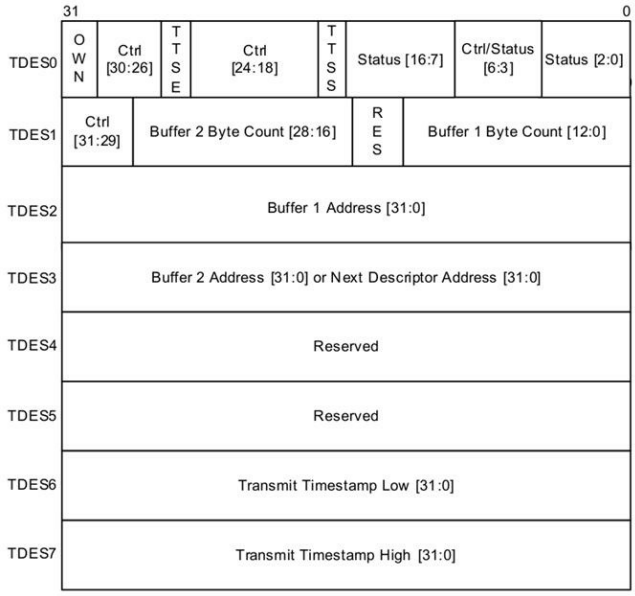
备用（增强）描述符结构

GMAC支持以下两种类型的描述符：普通描述符和增强描述符。 正常描述符可以有4个DWORDS（16字节），允许数据缓冲区最多2048字节。 增强描述符可以有8个DWORDS（32字节），已经实现支持高达8KB的缓冲区（对巨型帧有用）。 它们之间的主要区别在于TDES0，TDES1和RDES1中的控制和状态位的重新分配，并且核心还需要增强描述符来支持全IPC卸载和IEEE 1588时间戳，所以我们选择增强描述符 设备驱动程序。

发送描述符

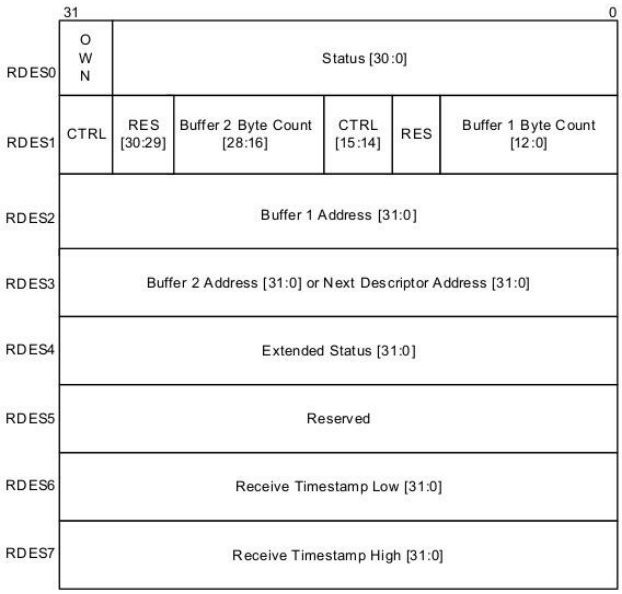
发送描述符结构如图3所示。设备驱动程序必须在描述符初始化期间对控制位TDES0 [31:18]进行编程。 当DMA更新描述符时，它将回写除OWN位（它清除）之外的所有控制位并更新状态位[7：0]。

通过提前支持时间戳，可以通过设置TDES0的位25（TTSE）为给定帧启用时间戳的快照。 当描述符关闭时（也就是说，当OWN位被清除时），时间戳被写入TDES6和TDES7。 这由TDES0的状态位17（TTSS）指示。



接收描述符

接收到的描述符的结构如图4所示。当我们使用Advanced Timestamp和IPC Full Offload功能时，它可以有32个字节的描述符数据（8个DWORD）。



编译

本节仅适用于Siflower提供的设备驱动程序。提供驱动程序包以及一个makefile，它允许将驱动程序编译为模块。运行make clean清理中间文件。

```
make clean
```

其他功能

PHY

GMAC的PHY芯片不在SF16A18芯片中，因此原则上可以为GMAC选择任何种类的PHY。 在GMAC设备驱动程序中，将从dts文件获取PHY的名称，然后找到PHY的驱动程序并使用它。 如果PHY没有驱动程序，则可以使用内核提供的通用PHY。

1588 Time Stamping

精确时间协议（PTP）包括IEEE 1588-2002标准（版本1）和IEEE 1588-2008（版本2），可以在测量和控制系统中精确同步时钟。 该协议支持异构系统，包括具有不同固有精度，分辨率和稳定性的时钟进行同步。

Linux为PTP提供了内核API。 这由ptp_clock_info结构表示。 下面给出了此设备接口的重要成员。

```
struct ptp_clock_info {

int (*adjfreq)(struct ptp_clock_info *ptp, s32 delta);
int (*adjtime)(struct ptp_clock_info *ptp, s64 delta);
int (*gettime)(struct ptp_clock_info *ptp, struct timespec *ts);
int (*settime)(struct ptp_clock_info *ptp, const struct timespec *ts);
int (*enable)(struct ptp_clock_info *ptp,
struct ptp_clock_request *request, int on);
...
}
```

设备驱动程序将其功能注册到Linux，如下所示。

```
static struct ptp_clock_info sgmac_ptp_clock_ops = {
        .adjfreq = sgmac_ptp_adjust_freq,
        .adjtime = sgmac_ptp_adjust_time,
        .gettime = sgmac_ptp_get_time,
        .settime = sgmac_ptp_set_time,
        .enable = sgmac_ptp_enable,
        ...
};
```

关机功能

GMAC支持断电功能，我们在驱动程序中完成了这个功能，但是我们并没有在Linux中选择它，所以驱动程序可以收到魔术数据包和电源管理中断，并将其报告给Linux，但内核中不会发生任何事情。

IP / TCP校验和卸载

GMAC支持IP和TCP / UDP / ICMP校验和卸载。
注意：要执行校验和卸载，需要在网络中支持。 在示例驱动程序中，我们利用Linux网络功能卸载硬件中的校验和计算。

Ethtool 命令

Linux向管理器网络设备提供ethtool命令。 这由platform_driver结构表示。该设备接口的重要成员在下面给出。

```
struct ethtool_ops {
        int     (*get_settings)(struct net_device *, struct ethtool_cmd *);
        int     (*set_settings)(struct net_device *, struct ethtool_cmd *);
        ...
}
```

设备驱动程序将其功能注册到Linux，如下所示。

```
static const struct ethtool_ops sgmac_ethtool_ops = {
        .get_settings = sgmac_ethtool_get_settings,
        .set_settings = sgmac_ethtool_set_settings,
        ...
};
```

我们测试ethtool的一些命令，并发现以下命令是有用的：

命令	描述
ethtool -s|--change DEVNAME	变更通用设置
ethtool -a|--show-pause DEVNAME	显示暂停选项
ethtool -A|--pause DEVNAME	设置暂停选项
ethtool -i|--driver DEVNAME	显示驱动信息
ethtool -d|--register-dump DEVNAME	dump寄存器
ethtool -r |--negotiate DEVNAME	重启协商
ethtool -S|--statistics DEVNAME	显示状态统计
ethtool --phy-statistics DEVNAME	显示phy状态统计
ethtool -T|--show-time-stamping DEVNAME	显示时间戳功能
ethtool -f|--flash DEVNAME	烧录镜像
ethtool -P|--show-permaddr DEVNAME	显示永久的硬件地址

注意：DEVNAME是ethxx，而xx表示Linux提供的接口名称。 见2.8注释。

驱动限制

这里列出了设备驱动程序软件的一些限制

> 不支持巨型帧处理。 我们没有测试它。
> 驱动程序的工作原理是收到的数据最多为8192字节。 如果收到更大的帧，驾驶员行为是未知的。