**网络驱动模块研究**

SWD 聂勇

**版 本 历 史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本/状态** | **责任人** | **起止日期** | **备注** |
| V1.0/正式 | 聂勇 | 22Nov2010 | BSP中网络驱动模块研究 |

**目录**

[1. 说明 3](#_Toc279066425)

[2. BSD4.3&SENS 3](#_Toc279066426)

[3. 初始化 4](#_Toc279066427)

[3.1 END设备驱动程序的初始化 4](#_Toc279066428)

[3.1.1 指定END设备 4](#_Toc279066429)

[3.1.2 装载END设备 4](#_Toc279066430)

[3.1.3 启动END设备 4](#_Toc279066431)

[4. 发送数据包 4](#_Toc279066432)

[4.1 数据结构的说明 5](#_Toc279066433)

[5. 接收数据包 6](#_Toc279066434)

# 说明

本文档是MIPS BSP中网络驱动模块的研究文档。主要记录在BSP的学习中，网络驱动部分遇到的相关问题。

下面主要以FTP建立和vxWorks镜像加载，探究网络驱动的实现和原理。



图 ‑1FTP建立流程图

# BSD4.3&SENS

为什么我们的网络协议驱动叫做END（Enhanced Network Driver）呢？下面从整体上描述一下vxWorks下的网络驱动。

首先，要了解的是vxWorks所支持的网络堆栈。vxWorks中支持了两种类型的网络堆栈：vxWorks BSD 4.3网络堆栈和vxWorks可裁剪的增强型网络堆栈（SENS,Scalable Enhanced Networks Statck）。vxWorks BSD 4.3协议栈符合BSD4.3标准（什么是BSD4.3标准？），提供了网络设备驱动程序与IP协议的紧密结合。所以，vxWorks BSD 4.3协议栈对应的网络驱动就是vxWorks BSD 4.3网络驱动。vxWorks可裁剪的增强型网络堆栈提供了可替换的网络设备驱动程序，所以，vxWorks可裁剪的增强型网络堆栈的驱动程序就称作增强的网络驱动程序（END,Enhanced Networks Driver）。

下面的图就形象说明了这个关系。



图 ‑1vxWorks网络协议栈和网络驱动

那么，为什么说“vxWorks可裁剪的增强型网络堆栈提供了可替换的网络设备驱动程序”呢？在具体的网络设备中，又是如何实现的呢？在分析END的时候，需要自己认真体会。

# 初始化

这里的初始化可能会涉及到很多关于vxWorks中的知识，然后才是BSP中网络驱动的部分。而《usrRoot()函数.docx》中的网络驱动部分更多的是BSP中部分的描述，请有选择性的阅读。

## END设备驱动程序的初始化

END设备驱动程序的初始化过程分为三个步骤来完成。即指定END设备，装载END设备，启动END设备。

### 指定END设备

### 装载END设备

### 启动END设备

# 发送数据包



## 数据结构的说明

这里需要特别说明的就是DMA中的数据结构，因为这里涉及到具体的外设地址。如果没有认真研究，很可能会被其混淆。

下面是DMA的信息结构体dma\_info\_t，位于最上层，包含了很多其他的结构体。

|  |
| --- |
| hnddma.c |
| typedef struct dma\_info {  ……  dmaregs\_t \*regs; /\* dma engine registers \*/  dmadd\_t \*txd; /\* pointer to chip-specific tx descriptor ring \*/  ……  }dma\_info\_t; |

有关DMA寄存器的结构体dmaregs\_t定义如下。

|  |
| --- |
| hnddma.h |
| typedef volatile struct {  /\* transmit channel \*/  uint32 xmtcontrol; /\* enable, et al \*/  uint32 xmtaddr; /\* descriptor ring base address (4K aligned) \*/  uint32 xmtptr; /\* last descriptor posted to chip \*/  uint32 xmtstatus; /\* current active descriptor, et al \*/  /\* receive channel \*/  uint32 rcvcontrol; /\* enable, et al \*/  uint32 rcvaddr; /\* descriptor ring base address (4K aligned) \*/  uint32 rcvptr; /\* last descriptor posted to chip \*/  uint32 rcvstatus; /\* current active descriptor, et al \*/  } dmaregs\_t; |

有关DMA解析表（descriptor table）的结构体dmadd\_t定义如下。

|  |
| --- |
| hnddma.h |
| typedef volatile struct {  uint32 ctrl; /\* misc control bits & bufcount \*/  uint32 addr; /\* data buffer address \*/  } dmadd\_t; |

那么，这些数据的值又分别是什么呢。我们以dma\_txfast()函数中的一段为例。

|  |
| --- |
| hnddma.c |
| int  dma\_txfast(dma\_info\_t \*di, void \*p0, uint32 coreflags)  {  ……  /\* init the tx descriptor \*/  W\_SM(&di->txd[txout].ctrl, BUS\_SWAP32(ctrl));  W\_SM(&di->txd[txout].addr, BUS\_SWAP32(pa + di->dataoffset));  ……  /\* kick the chip \*/  W\_REG(&di->regs->xmtptr, I2B(txout));  ……  } |

其实，dma\_info\_t中的dmaregs\_t \*regs;和dmadd\_t \*txd;已经是指向了寄存器和解析表的地址。也就是说，di->regs的值就是DMA的寄存器的物理地址的存储空间映射，也就是0xb8001200。同理，di->txd就是解析表在内存中的首地址，而di->txd[value]就是解析表的第value个条目。

# 接收数据包