Application Note

*如何将 **Linux** 驱动程序集成到您的系统中*

Alvaro Reyes

摘要

Linux 驱动程序是基本的软件组件，支持操作系统与硬件设备进行通信，例如显卡、打印机和以太网物理层设备 (PHY)。如果没有驱动程序，Linux 将无法有效地使用硬件，从而导致设备无法被识别或无法正常工作。本文档旨在为想要将 PHY 功能集成到 Linux 系统的开发人员提供全面的指导。本应用手册通过详细介绍在 Linux 内核中实现 PHY 驱动程序的复杂之处，为开发人员提供必要的知识和工具，以确保在各种硬件平台之间实现无缝集成、高性能和兼容性。

内容

商标

1 德州仪器 (TI) 以太网 PHY 驱动程序

所有以太网驱动程序（RTOS 和 Linux）均可在 TI 的以太网软件 GitHub 上找到。

表 1-1. TI 以太网 PHY 驱动程序

驱动器	支持的器件	U-boot 驱动程序	Linux 驱动程序
dp83822.c	DP83822	不适用	是，都可以在此处找到：TI 的以太网软件 GitHub
	DP83825	不适用	
	DP83826	不适用	
dp83848.c	DP83848	不适用	
	DP83620	不适用	
dp83867.c	DP83867	是	
dp83869.c	DP83869	是	
dp83tc811.c	DP83TC812	不适用	
dp83tc812.c	DP83TC812	不适用	
	DP83TC813	不适用	
	DP83TC814	不适用	
dp83tg720.c	DP83TG720	不适用	
	DP83TG721	不适用	

2 以太网 PHY 驱动程序概述

以太网 PHY Linux 驱动程序在实现网络接口控制器 (NIC) 与物理以太网介质之间的通信上发挥着至关重要的作用。这些驱动程序与 Linux 内核的网络子系统交互，为高级网络协议和应用提供了标准化接口。实现以太网 PHY 驱动程序涉及到自动协商、链路检测、速度和双工配置以及错误处理等处理任务。此外，这些驱动程序通常支持各种以太网标准，包括 10/100/1000Mbps 以太网。

图 2-1 是一个例子，说明了以太网 PHY 驱动程序的作用。从顶部开始，用户通过终端输入命令（例如 ethtool 命令）。ethtool 是一个 Linux 网络实用程序，接受用户在终端提供的输入并检查给定的参数是否有效。这是一个重要的步骤，它为用户提供高级接口与内核交互，而无需直接进行内核控制。如果参数正确，ethtool 会将命令传递给 MAC 和 PHY 驱动程序。这些驱动程序具有函数定义，以执行用户最初提供的命令并将这些命令应用到硬件。

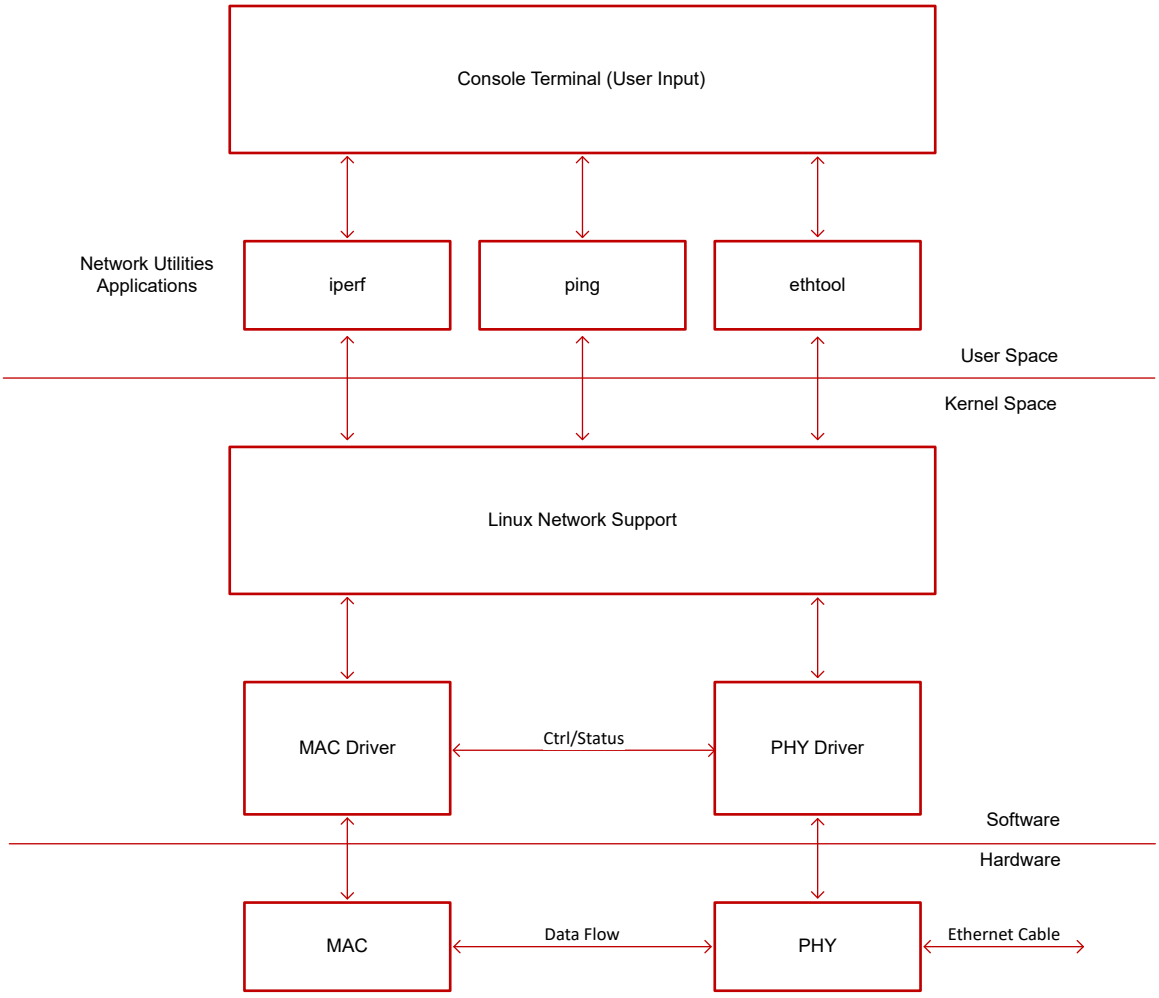

图 2-1. Linux 驱动程序方框图

2.1 了解 Linux 驱动程序类型

本应用手册讨论了 Linux 操作系统中两种主要的驱动程序类型：内核和 U-boot。

2.1.1 U-Boot 驱动程序

U-Boot 驱动程序是专为 U-Boot 引导加载程序设计的软件组件，通常用在嵌入式系统和自动加载过程中。这些驱动程序使得 U-Boot 能够在启动序列期间与各种硬件外设进行交互并控制这些硬件外设。这就在控制权移交给操作系统内核之前完成正确的系统初始化和硬件设置。

2.1.2 内核驱动程序

内核驱动程序也简称为*驱动程序*，它是一个软件组件，让操作系统内核能够与硬件设备进行通信并控制硬件设备。这些驱动程序集成到内核中，负责管理软件应用程序与硬件外设（如网络适配器、存储驱动器、输入或输出设备等）之间的交互。内核驱动程序负责处理设备初始化、数据传输、中断处理、电源管理和错误处理等任务。

3 驱动程序集成

将驱动程序集成到 Linux 系统中涉及多个关键步骤，以提供无缝的兼容性和功能性。最初，开发人员必须编译驱动程序代码以生成可加载的内核模块，或将代码直接整合到内核中（要在 MDIO 探测期间更快地识别 PHY，优先选择后者）。

以下各节更详细地描述了使用以下设置的这一过程。J721EXCPXEVM 通用处理器板与 J721EXSOMG01EVM TDA4VM 和 DRA829V 插槽式模块上系统配合使用。Linux-RT SDK 用于评估板上的 Linux 内核版本 5.10。通用处理器板本身有一个以太网端口，使用 DP83867E 以太网 PHY。子卡有另外的四个以太网端口，上面的插头插入通用处理器板的 EVM 扩展连接器，其中以太网 PHY 的驱动程序不包含在处理器的 SDK 中。

3.1 Linux 器件树

Linux 器件树是用于描述嵌入式系统中硬件组件和配置的数据结构。器件树为操作系统提供了一种标准化方式来了解硬件布局，包括有关处理器、内存、总线和外设的详细信息。器件树数据通常以二进制格式（**.dtb** 文件）存储，并在启动期间传递给 Linux 内核。然后，内核使用此信息来动态绑定器件树节点并初始化硬件组件，从而允许在不同的嵌入式平台上提供高效灵活的硬件支持，而无需将硬件详细信息硬编码到内核中。

器件树代码块示例说明了如何在器件树文件中配置子卡上的四个以太网 PHY。CPSW 是指处理器的 MAC 接口，要考虑的主要节点定义如下：

- &cpsw0 {}，初始化四个 RGMII 接口
- cpsw0_port*n* {}，初始化每个端口的更多详细信息
  - phy-mode：设置该端口的 MAC 接口
  - phy-handle：定义如何设置 PHY
    - <&cpsw9g_phy*x*> 用于设置 PHY 地址
      - 请注意，*x* 不设置 PHY 地址，只是一种命名约定。该地址在器件树代码块中分配到较低的位置，位于 cpsw9g_mdio{} 定义内部。
        - reg = <*x*>;
    - 这里也可以设置 RGMII 延迟，在 J721E 通用处理器板 dts 文件第 744 行和 RGMII 代码块中可以看到示例。
      - 通常，我们的 RGMII 延迟建议是，将 PHY 配置为 TX 和 RX CLK 延迟 2.0ns（称为*移位模式*），而处理器设置为 0 延迟（称为*对齐模式*）。有关更多信息，请参阅表 3-1。
        - 然而，许多 TI 处理器的 TX 线路内部有无法禁用的 2.0ns 延迟。因此在 RGMII 代码块中，PHY 上仅配置了 RX 延迟。
  - phys
    - 设置为每个端口分配的 eth#
      - <&cpsw0_phy_gmii_sel *n*>

表 3-1. RGMII 移位配置

MAC 配置	所需的 PHY 配置
在 RX 上对齐	在 RX 上移位
在 RX 上移位	在 RX 上对齐
在 TX 上对齐	在 TX 上移位
在 TX 上移位	在 TX 上对齐

RGMII 代码块：

```
&davinci_mdio {
    phy0: ethernet-phy@0 { //PHY0 is defined and passed to phy-handle
        reg = <0>;
        ti,rx-internal-delay = <DP83867_RGMIIDCTL_2_00_NS>;
        ti,fifo-depth = <DP83867_PHYCR_FIFO_DEPTH_4_B_NIB>;
    };
};

&cpsw_port1 {
    phy-mode = "rgmii-rxid";
    phy-handle = <&phy0>;
};
```

器件树代码块：

```
&cpsw0 {
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&mdio_pins_default
             &rgmii1_pins_default
             &rgmii2_pins_default
             &rgmii3_pins_default
             &rgmii4_pins_default
             >;
};

&cpsw0_port1 {
    phy-handle = <&cpsw9g_phy0>;
    phy-mode = "rgmii-rxid";
    mac-address = [00 00 00 00 00 00];
    phys = <&cpsw0_phy_gmii_sel 1>;
};

&cpsw0_port2 {
    phy-handle = <&cpsw9g_phy4>;
    phy-mode = "rgmii-rxid";
    mac-address = [00 00 00 00 00 00];
    phys = <&cpsw0_phy_gmii_sel 2>;
};

&cpsw0_port3 {
    phy-handle = <&cpsw9g_phy5>;
    phy-mode = "rgmii-rxid";
    mac-address = [00 00 00 00 00 00];
    phys = <&cpsw0_phy_gmii_sel 3>;
};

&cpsw0_port4 {
    phy-handle = <&cpsw9g_phy8>;
    phy-mode = "rgmii-rxid";
    mac-address = [00 00 00 00 00 00];
    phys = <&cpsw0_phy_gmii_sel 4>;
};
&cpsw9g_mdio {
    bus_freq = <1000000>;
    #address-cells = <1>;
    #size-cells = <0>;

    cpsw9g_phy0: ethernet-phy@0 {
        reg = <0>;
    };
    cpsw9g_phy4: ethernet-phy@4 {
        reg = <4>;
    };
    cpsw9g_phy5: ethernet-phy@5 {
        reg = <5>;
    };
    cpsw9g_phy8: ethernet-phy@8 {
        reg = <8>;
    };
};
```

当电路板正在运行时，可以使用终端命令 *dmesg grep | mdio* 确认 PHY 地址 (phy[*x*]) 和 eth 端口 (eth*n*)。

```
davinci_mdio c000f00.mdio: phy[0]: device c000f00.mdio:00, driver TI DP83TG720CS1.1
davinci_mdio c000f00.mdio: phy[4]: device c000f00.mdio:04, driver TI DP83TG721CS1.0
davinci_mdio c000f00.mdio: phy[5]: device c000f00.mdio:05, driver TI DP83TC812CS2.0
davinci_mdio c000f00.mdio: phy[8]: device c000f00.mdio:08, driver TI DP83TC814CS2.0
am65-cpsw-nuss c000000.ethernet eth4: PHY [c000f00.mdio:08] driver [TI DP83TC814CS2.0] (irq=POLL)
am65-cpsw-nuss c000000.ethernet eth3: PHY [c000f00.mdio:05] driver [TI DP83TC812CS2.0] (irq=POLL)
am65-cpsw-nuss c000000.ethernet eth2: PHY [c000f00.mdio:04] driver [TI DP83TG721CS1.0] (irq=POLL)
am65-cpsw-nuss c000000.ethernet eth1: PHY [c000f00.mdio:00] driver [TI DP83TG720CS1.1] (irq=POLL)
```

3.2 集成式驱动程序

本节介绍如何将驱动程序（newDriver.c，其中 newDriver 是以太网 PHY）添加到 Linux 系统上的 SDK 中，该系统缺少这一驱动程序或使用了过时的版本。

在 SDK 中，找到 Linux 内核目录 (LKD)。文件路径示例如下所示：

```
SDK_Install_Directory/board-support/TI-linux-kernel/
```

在该示例中，TI-Linux-kernel 是 LKD。从这里可以导航至：

```
LKD/drivers/net/phy/
```

将 newDriver.c 复制到该目录中。同一个目录中有 *Makefile* 和 *Kconfig*，这两个文件都需要编辑才能生成 newDriver.c。

编辑 Makefile

将以下代码行添加到 Makefile。请注意，赋值是 newDriver.o 而非 newDriver.c

```
obj-$(CONFIG_newDriver_PHY)     += newDriver.o
```

编辑 Kconfig

然后，将以下代码行添加到 Kconfig。

```
config newDriver PHY
    tristate "<Insert Company name> newDriver PHY"
    --help--
        Supports the newDriver PHY.
```

编辑 Makefile 和 Kconfig 文件后，返回到 LKD。从这里转到：

```
LKD/arch/arm64/configs
```

备注

如果您的处理器是 32 位而不是 64 位，请进入“arm”文件夹，而非“arm64”。

在此处可找到 defconfig 文件，添加以下代码行：

```
CONFIG_newDriver_PHY = y
```

命名约定 *CONFIG_newDriver_PHY* 需要与 Makefile 中设置的内容相匹配。

从这里可以返回到 SDK 安装目录并在终端上运行 *make* 命令。

备注

并非所有内核都可以通过运行 *make* 生成，请参阅 SDK 文档以了解生成内核、u-boot 和 dtb 文件的正确步骤。

4 常见终端命令

以下是几个命令及其说明、用例和终端输出。

4.1 初始化命令

系统启动后，以下命令说明驱动程序是否已正确加载。

4.1.1 dmesg | grep -i mdio

dmesg 是一个 Linux 命令，显示写入内核的消息。**|** 符号叫做 pipe 命令，它将一条命令的输出直接连接至另一个命令的输入。grep 是用于查找字符串的 Linux 命令，**-i** 参数忽略字符串的大小写。总之，**dmesg | grep -i mdio** 会查找所有写入内核的消息并筛选出包含 *mdio* 的消息。MDIO 接口是处理器访问 PHY 寄存器的方式。

该命令的目的是，确认驱动程序是否正确加载，或者从软件角度提供有关 PHY 行为错误起因的几条调试线索。

不良输出示例：

```
davinci_mdio c000f00.mdio: phy[10]: device c000f00.mdio:0a, MDIO device at address 10 is missing.
```

该消息指示未在 MDIO 总线上找到 PHY，这可能是由多个问题引起的。最常见的原因是器件树缺失或不正确（更多信息参阅节 3.1），但也可能是因为 PHY 无法正常工作或 MDIO 连接不良。

一旦可以在 MDIO 总线上检测到 PHY，另一个常见的错误消息是：

```
am65-cpsw-nuss c000000.ethernet eth1: PHY [c000f00.mdio:0a] driver [Generic PHY] (irq=POLL)
davinci_mdio c000f00.mdio: phy[10]: device c000f00.mdio:0a, driver unknown
```

驱动程序未知 和 *通用 PHY* 消息均指示驱动程序文件未正确加载、未编译或完全丢失；Linux 加载的通用驱动程序无法配合 PHY 正常运行。在这种情况下，请验证驱动程序是否已成功编译并添加到 Linux。更多有关该过程的信息，请参阅节 3.2。

最后，良好输出示例如下所示：

```
root@j7-evm:~# dmesg | grep mdio
davinci_mdio 46000f00.mdio: phy[0]: device 46000f00.mdio:00, driver TI DP83867
am65-cpsw-nuss 46000000.ethernet eth0: PHY [46000f00.mdio:00] driver [TI DP83867] (irq=POLL)
```

在此处我们可以看到，*phy[0]* 识别为 DP83867 并分配为端口 *eth0*

备注

PHY[*n*]，其中 *n* 表示 PHY 地址可以不同于 eth*x*，*x* 表示 PHY 分配到的端口。例如，在分配到端口 eth0 时，PHY 地址可以是 8。

4.1.2 ifconfig

ifconfig（区分大小写）是一个 Linux 终端命令，可显示网络接口，还可用于确定是否正确加载了驱动程序。**ifconfig -ethx down** 停用接口，而 **ifconfig -ethx up** 激活接口；这将再次加载驱动程序，类似电路板初次通电时。

```
root@j7-evm:~# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500  metric 1
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:f9:5b:d7:a4  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500  metric 1
        ether 34:08:e1:59:5c:d2  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500  metric 1
        ether d2:eb:75:2d:68:21  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536  metric 1
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 82  bytes 6220 (6.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 82  bytes 6220 (6.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

4.2 功能命令

本节中列出的命令提供一项服务或功能。

4.2.1 Phytool

Phytool 是用于访问 MDIO 寄存器的 Linux 命令，它提供了一种简单的方法来读取和写入 PHY 寄存器。该工具已经集成到 TI SDK 中，但可通过输入以下 *sudo apt-get install -y net-tools* 命令来下载。

Phytool 命令：

- *phytool read ethx/PHYADDRESS/desiredRegister*
- *phytool write ethx/PHYADDRESS/desiredRegister writeValue*

```
root@j7-evm:~# phytool read eth1/10/0x0    //Read Register 0x0
0x1140                                     //Result
root@j7-evm:~# phytool write eth1/10/0x0 0x0140     //Write Reg 0x0 = 0x0140
root@j7-evm:~# phytool read eth1/10/0x0             //Read again to confirm write command
0x0140
```

4.2.2 ethtool

ethtool 用于访问或更改网络驱动程序设置。

- *ethtool -ethx*
  - eth*x* 表示您指向的网络设备。如果电路板有两个以太网端口，则可以是 eth0 和 eth1
  - 命令可告知该以太网设备的当前状态和配置
    - 这是一种确认端口的 PHYADDRESS 的简单方法

```
ethtool eth0
Settings for eth0:
        Supported ports: [ TP    MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Link partner advertised link modes:  10baseT/Half 10baseT/Full
                                             100baseT/Half 100baseT/Full
                                             1000baseT/Full
        Link partner advertised pause frame use: Symmetric Receive-only
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Auto-negotiation: on
        master-slave cfg: preferred slave
        master-slave status: slave
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: external
        MDI-X: Unknown
        Supports Wake-on: ubgs
        Wake-on: d
        SecureOn password: 00:00:00:00:00:00
        Current message level: 0x000020f7 (8439)
                               drv probe link ifdown ifup rx_err tx_err hw
        Link detected: yes
```

4.2.3 强制执行主/从配置

此函数强制网络接口进入*主/从*状态，用户无需知道要写入哪些寄存器。这对于单线对以太网设备 (SPE) 尤其重要。在 SPE 通信中，需要有*主器件*和*从器件*，两者不能都是*主器件*或*从器件*。

在以下代码块中，使用 *ethtool* 命令检查 PHY 的当前状态，然后该状态更改为另一个状态并再次进行检查。

命令：

- ethtool -s eth*x* master-slave forced-master
- ethtool -s eth*x* master-slave forced-slave

备注

此命令没有输出。运行 *ethtool eth3 | grep master-slave* 以检查当前状态

```
root@j7-evm:~# ethtool eth3 | grep master-slave
        master-slave cfg: forced master
        master-slave status: master
root@j7-evm:~# ethtool -s eth3 master-slave forced-slave
root@j7-evm:~# ethtool eth3 | grep master-slave
        master-slave cfg: forced slave
        master-slave status: slave
root@j7-evm:~#
```

4.3 诊断命令

本节中列出的命令用于调试特定于以太网的应用问题。

4.3.1 SQI

信号质量指标 (SQI) 特性并非在所有 PHY 中都实现，它用于检查信号质量（**0** 表示无链路，**7** 表示最佳）。用于调试错误源头，例如：链路稳定但系统发现丢包。SQI 值较低，表示问题可能出在 MDI 接口（连接器侧）内部。如果器件的驱动程序中实现了 SQI，可以在 *ethtool* 命令中找到 SQI。如果只想显示 SQI，可以使用 *ethtool ethx | grep SQI*。

```
ethtool eth3
Settings for eth3:
        Supported ports: [ TP    MII ]
        Supported link modes:   100baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: No
        Supported FEC modes: Not reported
        Advertised link modes:  Not reported
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: No
        Advertised FEC modes: Not reported
        Speed: 100Mb/s
        Duplex: Full
        Auto-negotiation: off
        master-slave cfg: forced master
        master-slave status: master
        Port: Twisted Pair
        PHYAD: 5
        Transceiver: external
        MDI-X: Unknown
        Supports Wake-on: d
        Wake-on: d
        Current message level: 0x000020f7 (8439)
                               drv probe link ifdown ifup rx_err tx_err hw
        Link detected: yes
        SQI: 4/7
ethtool eth3 | grep SQI
        SQI: 4/7
```

4.3.2 TDR

时域反射计 (TDR) 是识别电缆故障的功能。并非所有以太网 PHY 都具有 TDR 功能，请务必检查 PHY 的数据表确认。要让 TI 的汽车单线对以太网 (SPE) PHY 正确运行 TDR，就必须知道 PHY 的*主/从*状态。

当 PHY 为*主器件*时：

- 如果电缆已连接（链路良好）
  - PHY 失掉链路、执行 TDR 并重获链路
- 如果电缆断开或损坏
  - PHY 执行 TDR 并输出：
    - 故障类型
      - 开路或短路
    - 故障距离（以米为单位）

当 PHY 为*从器件*时：

- 如果电缆已连接（链路良好）
  - 链路伙伴（*主器件*）需要强制静默（不传输任何数据包），否则 TDR 将失败
    - 电缆可与*主器件*断开，以使 TDR 作为*从器件*运行

在下面的代码块中，*eth3* 最初与已知良好的电缆相连并配置为*主器件*。TDR 按预期运行并完成，未检测到故障。TDR 完成后，从链路伙伴上拔下电缆，导致 *eth3：链路断开*。然后再次运行 TDR。

TDR 命令：*ethtool --cable-test ethx*

```
root@j7-evm:~# ethtool --cable-test eth3
am65-cpsw-nuss c000000.ethernet eth3: Link is Down
PHY is set as Master.
Cable test started for device eth3.
Cable test completed for device eth3.
Pair A code OK
TDR HAS COMPLETED AND PASSED
root@j7-evm:~# No Fault Detected.
am65-cpsw-nuss c000000.ethernet eth3: Link is Up - 100Mbps/Full - flow control off

am65-cpsw-nuss c000000.ethernet eth3: Link is Down

root@j7-evm:~# ethtool --cable-test eth3
PHY is set as Master.
Cable test started for device eth3.
Cable test completed for device eth3.
Pair A code Open Circuit
TDR HAS COMPLETED AND PASSED
Open Cable Detected
Length of Fault: 3 Meters
```

在下面的代码块中，***eth4*** 最初与已知良好的电缆相连并配置为*从器件*。TDR 按预期运行并失败。接下来，从链路伙伴上拔下电缆，导致 ***eth4*** *：链路断开*。然后再次运行 TDR。

```
root@j7-evm:~# ethtool --cable-test eth4
am65-cpsw-nuss c000000.ethernet eth4: Link is Down
PHY is set as Slave.
Cable test started for device eth4.
Cable test completed for device eth4.
TDR HAS FAILED
root@j7-evm:~# am65-cpsw-nuss c000000.ethernet eth4: Link is Up - 100Mbps/Full - flow control off

am65-cpsw-nuss c000000.ethernet eth4: Link is Down

root@j7-evm:~# ethtool --cable-test eth4
PHY is set as Slave.
Cable test started for device eth4.
Cable test completed for device eth4.
Open Circuit
TDR HAS COMPLETED AND PASSED
Open Cable Detected
Length of Fault: 3 Meters
```

4.3.3 吞吐量测试 - ping 和 iPerf

吞吐量测试是指将数据从一个板发送到另一个板。可首先执行初始 *ping* 测试，确认可与目标通信。以下示例展示了最基本的 ping 类型，其中主机直接连接到目标。此示例不包括任何类型的交换机、集线器或路由器。

ping 示例：运行 Linux 的测试板通过以太网电缆连接到 Linux PC。

1. 在 Linux PC 上，打开终端并运行 *ifconfig* 命令以查找 IPv4 地址
   a. （本例中为 169.254.132.246，标记为 *inet*）
2. 在测试板上，运行 *ifconfig 169.254.132.250*，以分配静态 IP 地址
   a. 请注意，该地址只有最后三位小数改变，才能是唯一的
      i. 称为主机 ID
   b. 地址的其余部分 (169.254.132) 必须相同
      i. 称为网络 ID
3. 从测试板运行 *ping 169.254.132.246*（Linux PC 的 IP 地址）
   a. ping 开始，可通过同时按下“ctrl”和“c”停止。

下面的代码块是从测试板 ping Linux PC 时捕获的，但 ping 可以双向执行，例如 Linux PC 也可以 ping 测试板。

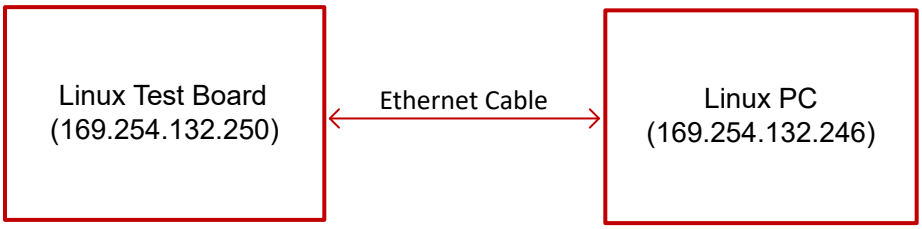

图 4-1. ping 方框图示例

```
root@j7-evm:~# ifconfig eth0 169.254.132.250
root@j7-evm:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  metric 1
        inet 169.254.132.250  netmask 255.255.0.0  broadcast 169.254.255.255
        inet6 fe80::3608:e1ff:fe59:5cd2  prefixlen 64  scopeid 0x20<link>
        ether 34:08:e1:59:5c:d2  txqueuelen 1000  (Ethernet)
        RX packets 133  bytes 16347 (15.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 481  bytes 117318 (114.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@j7-evm:~# ping 169.254.132.246
PING 169.254.132.246 (169.254.132.246): 56 data bytes
64 bytes from 169.254.132.246: seq=0 ttl=64 time=0.579 ms
64 bytes from 169.254.132.246: seq=1 ttl=64 time=0.546 ms
64 bytes from 169.254.132.246: seq=2 ttl=64 time=0.587 ms
64 bytes from 169.254.132.246: seq=3 ttl=64 time=0.557 ms
64 bytes from 169.254.132.246: seq=4 ttl=64 time=0.518 ms
64 bytes from 169.254.132.246: seq=5 ttl=64 time=0.574 ms
64 bytes from 169.254.132.246: seq=6 ttl=64 time=0.548 ms
64 bytes from 169.254.132.246: seq=7 ttl=64 time=0.561 ms
^C
--- 169.254.132.246 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0.518/0.558/0.587 ms
root@j7-evm:~#
```

ping 成功运行后，我们可以尝试使用 *iPerf*（一种用于测量网络性能/带宽的开源工具）执行吞吐量测试。*iPerf* 需要安装在两台计算机（测试板和 Linux PC）上才能正常运行。

iPerf 示例：

1. 在测试板上，运行命令 *iperf -s* 将测试板配置为服务器。
2. 在 Linux PC 上，运行命令 *iperf -c 169.254.132.250*（服务器的 IP 地址），将 Linux PC 配置为客户端并连接到服务器。

下面的代码块是从测试板捕获的。这里我们可以看到已成功传输 1.09GB 数据，带宽非常接近网络端口的广播速度 (1000Mbps)。

```
root@j7-evm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 169.254.132.250 port 5001 connected with 169.254.132.246 port 37356
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.09 GBytes   933 Mbits/sec    //This step happens after the Linux PC connects
as a client
```

5 总结

本应用手册全面概述了 Linux PHY 驱动程序基本术语，指导用户将新的以太网驱动程序集成到系统中并深入介绍了调试用的常见终端命令。从基础概念到实际实施，该资源为开发人员提供了必需的知识和工具以高效应对 Linux 驱动程序开发的复杂性。

6 参考资料

- 德州仪器 (TI)，*ti-ethernet-software Github*。
- ethtool，*Linux 手册页*。
- iPerf，*速度测试工具*。

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265