UM-BUS 总线网卡的驱动程序设计与实现*

张 霖[1],周继芹[1,2],张伟功[1,3]
(1.首都师范大学 信息工程学院,北京 100048;2.首都师范大学 数学科学学院,北京 100048;
3.高可靠嵌入式系统技术北京市工程研究中心,北京 100048)

摘 要:动态可重构高速串行总线(UM-BUS)具有多通道并发冗余的特征以及远程存储访问能力,可为 CPS、物联网中底层传感器与执行单元的远程接入提供一种高速可靠的解决方案。设计实现了一种基于 UM-BUS 总线连接的以太网卡的 Linux 驱动程序,解决了通过 PCIe、UM-BUS 两种总线进行设备跨总线访问的难题,实现了 UM-BUS 总线连接的以太网设备的驱动操作。使用第三方网络通信软件进行文件传输测试,网络通信功能正常稳定,通信速率与标准 PCIe 网卡基本一致,满足了 CPS 系统中通过 UM-BUS 总线上连接的以太网设备与外部系统进行数据通信的需求。

关键词:UM-BUS;网卡驱动;Linux 系统;跨总线驱动
中图分类号:TP393 **文献标识码**:A DOI:10.16157/j.issn.0258-7998.200143

中文引用格式:张霖,周继芹,张伟功. UM-BUS 总线网卡的驱动程序设计与实现[J].电子技术应用,2020,46(5):83-87.
英文引用格式:Zhang Lin,Zhou Jiqin,Zhang Weigong. Design and implementation of the driver for network card plugged via UM-BUS[J]. Application of Electronic Technique,2020,46(5):83-87.

Design and implementation of the driver for network card plugged via UM-BUS

Zhang Lin[1],Zhou Jiqin[1,2],Zhang Weigong[1,3]
(1.College of Information Engineering,Capital Normal University,Beijing 100048,China;
2.School of Mathematical Sciences,Capital Normal University,Beijing 100048,China;
3.Beijing Engineering Research Center of High Reliable Embedded System,Capital Normal University,Beijing 100048,China)

Abstract:Dynamic reconfigurable high-speed serial bus(UM-BUS) has the characteristics of multi-channel concurrent redundancy and the ability of remote storage access. It can provide a high-speed and reliable solution for the remote access of sensors and execution units in CPS and Internet of things. This paper designs and implements a driver of network card based on UM-BUS connection, solves the problem of device access through PCIe and UM-BUS multiple buses, and realizes the driver operation of Ethernet device connected by UM-BUS. In the Linux system, the third-party network communication software is used for file transmission test. The network communication function is normal and stable, and the communication rate is basically the same as that of the standard network card, which meets the requirements of data communication between the Ethernet equipment and the external through UM-BUS in the CPS system.

Key words:UM-BUS;Ethernet driver;Linux;cross bus driver

0 引言

动态可重构高速串行总线(UM-BUS)是针对嵌入式系统小型化与一体化设计提出的一种具备远程扩展能力和动态容错特征的高速串行总线[1],可以解决 CPS[2] 以及物联网在异构接入、动态连接、可靠性、实时性等方面的问题。在基于 UM-BUS 总线的 CPS 应用系统中,不仅需要通过总线本地接入传感器进行数据交互,还需要实现与其他 CPS 系统以及云服务之间的通信,因此,需要通过以太网与其他 CPS 系统进行通信从而构建更加全面完善的 CPS 系统。

在基于 UM-BUS 总线的 CPS 应用系统中,PC 通过 PCIe 总线与 UM-BUS 总线主控制器相连,所以在进行以太网通信时,PC 需要跨过 PCIe 总线和 UM-BUS 总线去驱动以太网,但现有的通用网卡驱动程序不能解决这个问题,因此提出一种基于 UM-BUS 总线的网卡驱动去解决跨两种总线进行设备访问的问题。

*基金项目:国家自然科学基金(61772350);共用信息系统装备预先研究项目(公开)(JZX2017-0988/Y300);北京市科技新星计划(Z181100006218093);体系结构国家重点实验室开放课题(CARCH201607);北京未来芯片技术高精尖创新中心科研基金资助项目(KYJJ2018008);科技创新服务能力建设-基本科研业务费(科研类)(006-19530050173)

1 以太网 MAC 控制器介绍

1.1 以太网 MAC 控制器连接方式

在基于 UM-BUS 总线的 CPS 应用系统中，以太网 MAC 控制器的连接方式如图 1 所示，其中所使用的以太网是百兆网，PC 通过 PCIe 总线与 UM-BUS 主控制器相连，UM-BUS 主从控制器通过 UM-BUS 总线相连，UM-BUS 从控制器与以太网 MAC 控制器连接。

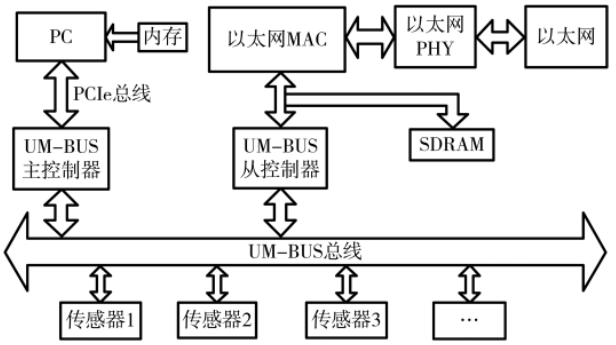

图 1 基于 UM-BUS 总线的 CPS 系统结构图

PC 采用 Linux 操作系统，通过以太网网卡驱动实现以太网传输，其中以太网 MAC 控制器是基于描述符链表的 MAC 控制器。SDRAM 用于存储描述符表和数据，PC 将内存中的数据通过 PCIe 总线和 UM-BUS 总线放入 SDRAM 中，以太网 MAC 也可以将数据放入 SDRAM 中。总线设备操作是通过 PCIe 总线和 UM-BUS 总线设置相关寄存器并启动相应的操作。在发送数据包时，以太网 MAC 控制器读取描述符并进行解析，拿到数据后通过以太网发出去；在接收数据包时，以太网 MAC 控制器根据描述符表找到对应的地址，将数据放入 SDRAM 中并产生中断，PC 响应中断进行相应处理。

1.2 以太网 MAC 控制器的描述符表

以太网 MAC 控制器是基于描述符链表进行数据传输，由发送控制器和接收控制器组成。图 2 所示的描述符表有 1 KB 描述符存储区，描述符分为发送描述符和接收描述符。一个描述符共两个 32 位字，占用 8 字节，因此描述符表最多可以存储 128 个描述符。每一个描述符包含一个控制字和一个指针，指针指向对应的数据包，每个数据包为 2 k。

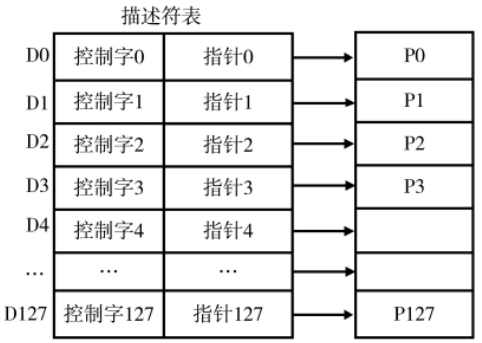

图 2 以太网 MAC 控制器的描述符表结构图

在描述符的控制字中，位 0~10 为传送长度，长度不大于 1 514。位 11 为传输使能位，一旦置位，启动本描述符的传输，传输完成后，会自动清 0。位 12 表示回卷，为 1 时，在完成本描述符后，描述符指针回到 0；为 0 时，描述符指针自动加 8 指向下一描述符。位 13 为中断使能位，为 1 时，如果控制寄存器中对应的中断使能标志有效，在本描述符处理完成后，不论传输是否正确，都会产生一个中断请求；为 0 时不产生中断。位 14~18为错误位，由 MAC 设置。

1.3 以太网 MAC 控制器的相关寄存器

以太网 MAC 控制器有四个相关寄存器，分别是 MAC 控制寄存器、MAC 状态与中断源寄存器、发送描述符表寄存器以及接收描述符表寄存器。

MAC 控制寄存器的作用是使能中断以及允许接收和发送，地址为 0x2000，其中位 6 为软件复位；位 3 为接收中断使能位；位 2 为发送中断使能位；位 1 为允许接收位；位 0 为允许发送位。

MAC 状态与中断源寄存器的主要作用是判断当前以太网数据包的传输状态，地址为 0x2016，其中位 5 为 1 表示在以太网包发送过程中，从存储器读取数据时总线通信错；位 4 为 1 表示在以太网包接收过程中，向存储器写入数据时总线通信错；位 3 为正确地发送了一个以太网包；位 2 为正确地接收到了一个以太网包；位 1 为以太网包发送时遇到了错误，发送中止；位 0 为以太网包接收时发生了错误，接收中止。

发送描述符表寄存器地址为 0x206e，包含发送描述符表基地址以及发送描述符表计数器。

接收描述符表寄存器地址为 0x2084，包含接收描述符表基地址和接收描述符表计数器。

2 基于 UM-BUS 总线的以太网网卡驱动总体设计

2.1 UM-BUS 总线驱动程序的体系结构

根据图 1，UM-BUS 总线的驱动程序应具有两种功能：(1)CPS 系统中的主控设备在加载总线驱动后，可通过 UM-BUS 访问接口对其他总线设备的地址空间进行读写访问，从而控制各从设备连接的传感器和执行器；(2)CPS 系统中的主控设备在加载总线驱动后，实现了跨 PCIe 和 UMBUS 总线访问，通过 UM-BUS 总线的以太网接口，实现云服务器与 CPS 系统之间的交互。

UM-BUS 总线设备驱动层主要包含三个模块：PCIe 模块、UM-BUS 总线模块以及 UM-BUS 总线网卡模块。UM-BUS 总线设备驱动层总体设计如图 3 所示。

PCIe 模块实现对总线设备媒介层的 PCIe 设备访问，UM-BUS 扩展板卡通过 PCIe 接口接入主机，上层应用程序发起的对以太网的访问以及对 UM-BUS 总线的访问最终都通过 PCIe 模块传递给硬件设备；UM-BUS 总线模块实现对 UM-BUS 总线设备的访问，包括对 UM-BUS 总线存储空间、IO 空间和属性空间的读写访问；UM-BUS 总线网卡模块实现对以太网网卡的控制，

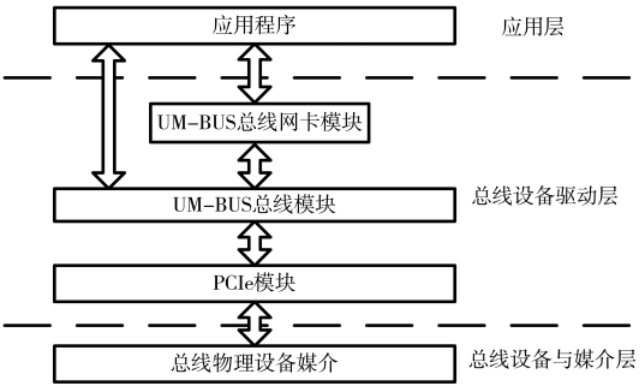

图 3 UM-BUS 总线设备驱动程序结构框图

通过加载标准的以太网驱动，应用层可以通过该网卡与其他主机进行通信。

2.2 UM-BUS 总线网卡驱动程序的体系结构

Linux 的网络系统主要是基于 BSD UNIX 的 socket 机制，在系统和驱动程序之间定义有专门的数据结构(sk_buff)进行数据的传递[3]，操作系统支持对发送数据和接收数据的缓存，提供流量控制机制，提供对多协议的支持[4]。Linux 系统对网络设备驱动定义为了 4 个层，从上往下分别为网络协议接口层、网络设备接口层、网络驱动接口层以及设备媒介层[5]。Linux 网络设备驱动程序结构如图 4 所示[6]。

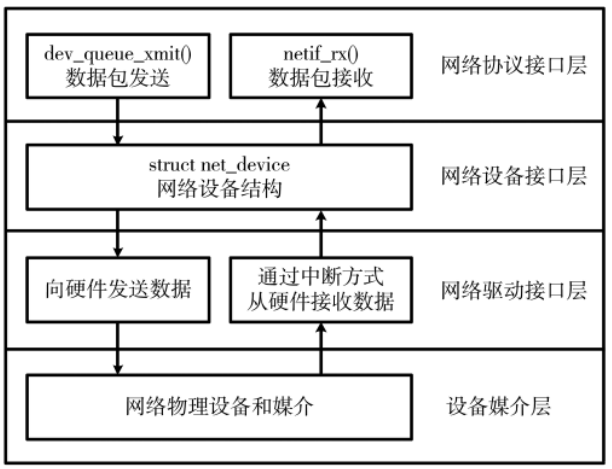

图 4 Linux 网卡驱动层次结构图

网络协议接口层是实现统一的数据包收发的协议，向网络协议提供统一的数据包收发接口，该层主要负责调用 dev_queue_xmit()函数发送数据包，以及 netif_rx()函数接收数据包，使用 sk_buff 作为数据的载体。

网络设备接口层是整个网络接口的关键部分，通过 net_device 网络设备结构体来描述一个具体的网络设备的信息，定义发送和接收函数，实现不同硬件的统一，并将 net_device 注册到内核[7]。

网络驱动接口层负责驱动网络设备硬件来完成各个功能，各个操作函数都是 net_device 数据结构的具体成员。

UM-BUS 总线网卡驱动程序需要完成的主要工作是网卡设备的初始化、网卡设备的发送和接收以及对中断的处理。

2.3 UM-BUS 总线设备访问

UM-BUS 总线控制器中设置了目标节点号寄存器、目标地址寄存器、访问数据寄存器、总线通信命令寄存器、DMA 传输控制寄存器以及总线状态寄存器等。UM-BUS 总线的设备访问是通过操作 UM-BUS 总线控制器的相关寄存器实现的，总线访问分为 UM-BUS 总线的单字、IO 以及 DMA 访问。

UM-BUS 总线的单字和 IO 访问过程：通过 PCIe 访问 UM-BUS 总线控制器的相关寄存器，向相应寄存器中写入目标节点号和目标地址，若为写操作，还需要写入访问数据，然后写入访问命令，启动访问，若为读操作，则需要读取返回的数据，通过 PCIe 访问总线状态寄存器来判断是否完成总线访问操作。

DMA 访问过程：初始化相关寄存器，通过 DMA 传输控制寄存器，控制 PC 内存到以太网数据缓冲区之间的跨总线 DMA 传输。

在通过以太网网卡发送数据包时，数据包从主机内存中发出，通过 PCIe DMA 存入 UM-BUS 控制器存储缓冲区，然后通过总线 DMA 将数据放入以太网网卡本地缓冲区，通过总线单字写操作修改发送描述符表中相应的控制字，从而将数据包发送出去。根据发送描述符的控制字位 0 是否为 0，判断当前是否发送完成。具体流程如图 5 所示。

在通过以太网网卡接收数据包时，首先，通过总线单字读操作读取接收描述符控制字判断以太网网卡是否接收到数据包，若接收到数据包，将数据包放入以太网网卡本地缓冲区，通过总线 DMA 将数据包放入 UM-BUS 控制器存储缓冲区，通过总线单字写操作将接收描述符控制字修正，重新使能该描述符进行后续接收，并产生中断进行相应处理，最后通过 PCIe DMA 将数据包放入主机内存。具体流程如图 6 所示。

2.4 UM-BUS 总线网卡驱动程序的实现

2.4.1 网卡设备的初始化

网卡设备的初始化函数是网卡驱动程序执行的第一步，该函数负责对以太网控制器进行初始化，分配及初始化 net_device 对象，向系统注册网络设备。

图 5 发送数据包时总线访问流程图

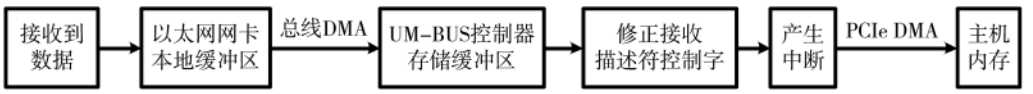

图 6 接收数据包时总线访问流程图

首先，通过 alloc_etherdev()函数分配一个 net_device 结构体，然后设置 net_device 结构体成员中的发包函数[8]，完成对以太网控制器的初始化，最后通过 register_netdev()函数将网卡设备注册到系统中从而完成网卡驱动程序的初始化工作。

以太网控制器初始化流程为：首先初始化 MAC 控制字，对 PHY 进行初始化操作，包括读取 PHY 标识寄存器、复位 PHY、配置 PHY 工作模式、启动自动协商、生成 MAC 控制字，然后设置 MAC 地址以及控制字，通过总线DMA写操作清空发送描述符缓冲区和接收描述符缓冲区，通过总线单字写操作设置 MAC 控制字，然后初始化以太网接收缓冲区描述符表，通过总线 DMA 写操作将所有构造好的描述符写入板上接收描述符缓冲区，最后启动 MAC 接收。

2.4.2 网卡数据包的发送

sk_buff 结构体是一个双向链表，含义为“套接字缓冲区”，用于网络各个层次之间传递数据，管理和控制接收或发送数据包的信息。上层通过 sk_buff 将数据包传递给驱动程序中的发包函数，发包函数负责将以太网包通过 UM-BUS 总线传送到总线控制器的存储器中，再通过以太网控制器发送出去。在进行数据包发送前先关闭中断，发送结束后再打开中断。具体流程如图 7 所示。

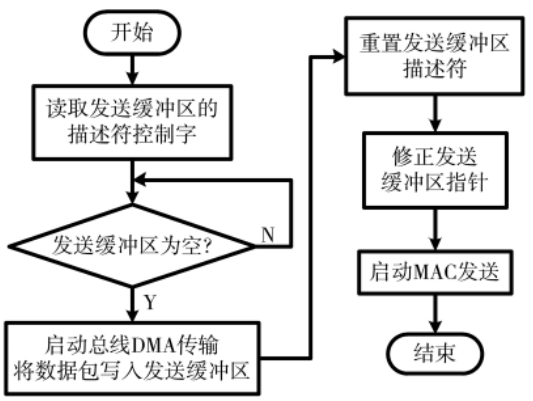

图 7 以太网控制器发包函数流程图

以太网控制器发送数据包流程为：

(1)读取发送缓冲区的描述符控制字；

(2)若位 11 为 1，说明发送缓冲区已满，需要等待当前描述传输完成。返回步骤(1)；

(3)若位 11 为 0，说明发送缓冲区为空，往发送缓冲区写入新的数据和描述符，先写入缓冲区地址字，写入完成后，修正发送缓冲区指针，然后写入描述符控制字，完成描述符写入后，启动 MAC 发送；

(4)读取 MAC 控制字，看是否正在发送，若位 0 为 0，表示目前 MAC 未处在发送状态，那么发送本次写入缓冲区的数据。将本次写入的描述符的地址写入 MAC 的发送描述符表寄存器，然后将 MAC 控制字位 0 置 1，启动当前描述符块的发送；

(5)若 MAC 控制字位 0 为 1，表示目前 MAC 正在发送过程中，本次加入的发送包已加入到发送链表中，会按其在缓冲区中的顺序自动发送；

(6)一个数据包的发送已成功启动，正常退出。

以上操作共进行 8 次 UM-BUS 总线设备访问。

2.4.3 网卡数据包的接收

网卡设备接收数据的主要方法是由中断引发设备的中断服务程序，中断服务程序对中断类型进行判断，如果是以太包的接收中断则进入该以太网控制器收包函数进行以太网包的接收，然后通过 DMA 将以太网包从接收缓冲区传输到主机内存。主机收到以太网包后，会构造一个 sk_buff 将数据放进去，再通过 netif_rx()函数提交给上层。以太网控制器收包函数的流程如图 8 所示。

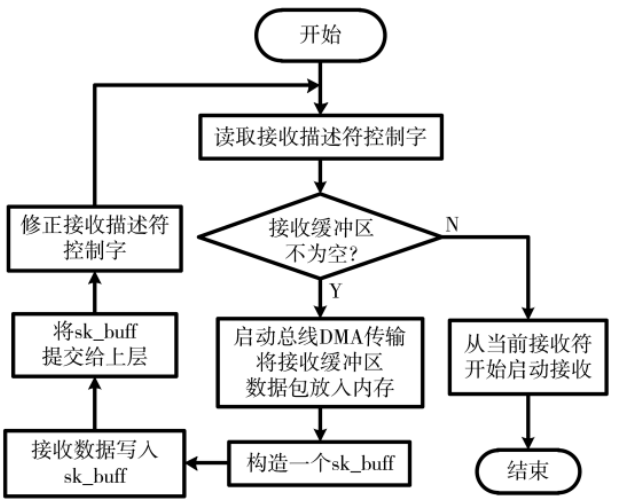

图 8 以太网控制器收包流程图

以太网控制器接收数据包流程为：

考虑到收发包的异步性，利用硬件的缓冲开辟了 8 个接收缓冲区构成了一个循环链表，当到最后一个描述符时需要回卷到第一个，指针序号要回到 0。

(1)读取接收缓冲区的描述符控制字；

(2)若位 11 为 0，表示该描述符对应的缓冲区已收到一个数据包，通过 DMA 把数据搬移到用户缓冲区，构造一个 sk_buff，将数据写入 sk_buff 中并提交给上层，将描述符控制字修正，重新使能该描述符进行后续接收，并产生中断进行相应处理，修正缓冲区指针序号，达到第 7 个时，需要回到 0；

(3)若位 11 不为 0,表示没有收到数据包,退出循环;

(4)若已经停止接收,需要重新启动接收,读取 MAC 控制字;

(5)若位 1 为 0,表示已停止接收,要重新启动接收,将接收描述符起始地址写到接收描述符表寄存器,然后将 MAC 控制字位 1 置 1,启动当前描述符块的接收。返回步骤(1)。

以上操作共进行 6 次 UM-BUS 总线设备访问。

3 驱动程序的测试

3.1 测试环境

为了验证 UM-BUS 总线网卡驱动的正确性和有效性,测试系统由一台装有 UM-BUS 总线扩展卡的 PC 器、一个 UM-BUS 总线扩展卡以及一台标准 PC 组成,通过 UM-BUS 总线连接。测试环境所使用的操作系统均为 Linux。

3.2 网卡驱动程序测试

3.2.1 连通性测试

按照测试环境的组织方式连接成功后,将编写好的驱动程序编译进内核并执行,执行 ifconfig 指令查看相关的以太网网卡信息,通过执行 ping 命令,两边是可以互通的。具体如图 9 所示[9]。

```
root@zhou:/home/zhou/Desktop/vnet-init-1# ping -I vnet_2 192.168.1.18
PING 192.168.1.18 (192.168.1.18) from 192.168.1.15 vnet_2: 56(84) bytes of data.
64 bytes from 192.168.1.18: icmp_seq=1 ttl=64 time=0.834 ms
64 bytes from 192.168.1.18: icmp_seq=2 ttl=64 time=0.350 ms
64 bytes from 192.168.1.18: icmp_seq=3 ttl=64 time=0.483 ms
```

图 9 测试结果图

采用第三方软件飞鸽传书进行传输文件测试,传输文件大小为 2 GB,传输 100 次,没有出现丢包情况;采用 HTTP 浏览网页功能均正确。

3.2.2 传输性能测试

进行 UDP 通信,使用 wireshark 抓包工具进行监测,主控设备 IP 地址为 192.168.1.18,目标设备 IP 地址为 192.168.1.15。具体如图 10 所示。

No.	Time	Source	Destination	Protocol	Length	Info
1607…	2231.1221685…	192.168.1.15	192.168.1.18	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1227504…	192.168.1.15	192.168.1.18	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1233741…	192.168.1.15	192.168.1.18	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1239590…	192.168.1.15	192.168.1.18	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1245597…	192.168.1.15	192.168.1.18	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1262371…	192.168.1.18	192.168.1.15	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1264732…	192.168.1.18	192.168.1.15	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1264814…	192.168.1.18	192.168.1.15	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1267369…	192.168.1.18	192.168.1.15	UDP	1502	49153 → 49153 Len=1460
1607…	2231.1267446…	192.168.1.18	192.168.1.15	UDP	1502	49153 → 49153 Len=1460

图 10 UDP 通信测试结果图

UDP 包传输情况如表 1 所示。传输文件为 10 GB,平均传输速率为 3.787 MB/s,没有出现丢包情况,传输过程中未发现数据错误。

表 1 传输情况统计表

传输文件大小/GB	平均传输速度/(MB/s)	丢包情况	错误情况
10	3.787	0	0

4 结论

本文提出一种适用于基于 UM-BUS 总线连接的以太网网卡驱动程序的设计方法,能够实现跨多种总线进行设备访问,在百兆网情况下,传输速率可以达到 3.787 MB/s,从而实现基于 UM-BUS 总线的 CPS 系统与其他 CPS 系统和云服务互联。

参考文献

[1] 张伟功,周继芹,李杰,等.UM-BUS 总线及接入式体系结构[J].电子学报,2015,43(9):1776-1785.

[2] 李仁发,谢勇,李蕊,等.信息-物理融合系统若干关键问题综述[J].计算机研究与发展,2012,49(6):1149-1161.

[3] 罗革新,张中杰,孙仕胜,等.基于 Linux 虚拟网卡技术构建数传电台 TCP/IP 通信平台的研究[J].电子技术与软件工程,2015(7):38-39.

[4] 袁安富,夏生凤.基于 ARM 和 Linux 的 DM9000 网络接口设计及驱动实现[J].计算机工程与科学,2011,33(2):27-31.

[5] 常锋,孟传良.基于 ARM-Linux 的网络驱动程序设计[J].通信技术,2012(6):36-39.

[6] 龙新辉,陈俊杰.基于嵌入式 Linux 的以太网卡驱动设计与实现[J].舰船电子工程,2011(3):149-152.

[7] 谢红薇,宋春燕.嵌入式 Arm-Linux 系统的网卡驱动程序的分析与实现[J].电脑开发与应用,2012(4):54-57.

[8] 张洪,吴钦章,杜春蕾.基于 Linux 虚拟网卡测试平台的系统设计[J].电子设计工程,2016,24(17):96-97.

[9] 高嵩,纪超,陈超波.基于嵌入式 Linux 的 DM9000 网络驱动设计[J].计算机与数字工程,2013(2):156-158.

(收稿日期:2020-02-24)

作者简介:

张霖(1996-),女,硕士研究生,主要研究方向:高可靠嵌入式系统。

周继芹(1978-),女,博士研究生,主要研究方向:计算机系统体系结构与性能评价。

张伟功(1967-),通信作者,男,研究员,博士生导师,主要研究方向:嵌入式系统体系结构、高能效计算、高可靠通信总线。

(上接第 82 页)

[10] 刘慧娟.基于蚁群算法的多飞行器协同目标搜索航迹规划[D].武汉:华中科技大学,2014.

[11] 向敏,牛立强,武沛羽,等.复杂电磁环境下无人机的雷达散射特性研究进展[J].电子技术应用,2019,45(6):1-6,10.

(收稿日期:2019-09-17)

作者简介:

吕洁(1988-),女,硕士研究生,讲师,主要研究方向:无线电信号、人工智能。

版权声明