Design and Implementation of the Driver for Network Card Plugged via UM-BUS

Lin Zhang[1,a] and Weigong Zhang[1,b]

[1]*College of Information Engineering, Capital Normal University, Beijing, China*
a. 2171002034@cnu.edu.cn, b. zwg771@cnu.edu.cn

Keywords: UM-BUS, Ethernet driver, Linux, cross bus driver.

Abstract: The 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 Linux driver of Ethernet card based on UM-BUS connection, solves the problem of cross bus access through PCIe and UM-BUS, and realizes the drive operation of Ethernet device connected by UM-BUS. Using the third-party network communication software for file transmission test, the network communication function is normal and stable, and the communication rate is basically the same as the standard PCIe network card, which meets the data communication requirements between the Ethernet equipment connected on the UM-BUS and the external system in the CPS system.

1. Introduction

Dynamic reconfigurable high-speed serial bus (UM-BUS) is a kind of high-speed serial bus with remote extension ability and dynamic fault-tolerant characteristics, which is proposed for the miniaturization and integration design of embedded system [1]. It can solve the problems of CPS [2] and Internet of things in heterogeneous access, dynamic connection, reliability, real-time and so on. In the application system of CPS based on UM-BUS, it is not only necessary to connect sensors locally through the bus for data interaction, but also to realize the communication with other CPS systems and cloud services. Therefore, we need to communicate with other CPS systems through Ethernet to build a more comprehensive CPS system.

In the CPS application system based on UM-BUS, the PC is connected with the main controller of UM-BUS through the PCIe bus, so in the Ethernet communication, the PC needs to drive the Ethernet across the PCIe bus and UM-BUS, but the existing universal network card driver cannot solve this problem, so we propose a network card driver based on UM-BUS to solve the problem of device access across the two buses.

2. Introduction to Ethernet MAC Controller

2.1.Connection Method of Ethernet MAC Controller

The Ethernet used in the CPS application system based on UM-BUS shown in Figure 1 is 100m network. The PC is connected with the UM-BUS master controller through the PCIe bus, the UM-BUS master-slave controller is connected with the UM-BUS, and the UM-BUS slave controller is connected with the Ethernet MAC controller.

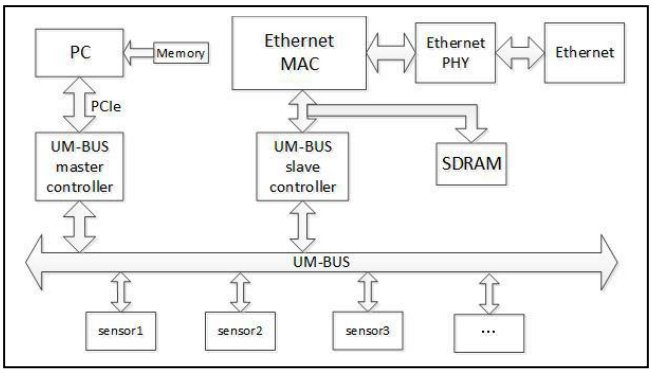

Figure 1: The structure diagram of CPS system based on UM-BUS.

The PC adopts the Linux operating system, and realizes the Ethernet transmission through the Ethernet network card driver. The Ethernet MAC controller is the MAC controller based on the descriptor chain table. SDRAM is used to store descriptors and data. PC puts the data in memory into SDRAM through PCIe and UM-BUS, and Ethernet MAC can also put the data into SDRAM. Bus device operation is to set relevant registers and start corresponding operations through PCIe and UM-BUS. When transmitting packets, the Ethernet MAC controller reads the descriptors and analyzes them, and transmits them through Ethernet after getting the data; when receiving packets, the Ethernet MAC controller finds the corresponding address according to the descriptor table, puts the data into SDRAM and generates an interrupt, and the PC responds to the interrupt and processes it accordingly.

2.2.Description Table of Ethernet MAC Controller

Ethernet MAC controller is based on descriptor chain table for data transmission, which is composed of transmitting controller and receiving controller. The descriptor table shown in Figure 2 has a 1KB descriptor storage area. One descriptor has two 32-bit words, occupying 8 bytes. Therefore, the descriptor table can store up to 128 descriptors. Each descriptor contains a control word and a pointer. The pointer points to the corresponding packet, and each packet is 2K.

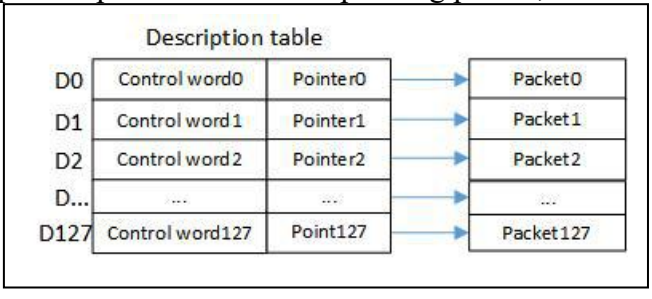

Figure 2: Structure chart of descriptor table of Ethernet MAC controller.

In the control word of the descriptor, bits 10-0 are the transmission length, in bytes, and the length is not greater than 1514; bit 11 is the transmission enabling bit, once set, the transmission of the descriptor will be started, and after the transmission is completed, it will automatically clear 0; bit 12 means rewinding, when it is 1, the descriptor pointer will return to 0 after the completion of the descriptor; when it is 0, the descriptor pointer will automatically add 8 to point to the next descriptor, but add When it reaches the 1K boundary, it will automatically return to 0; bit 13 is the interrupt enable bit, when it is 1, if the corresponding interrupt enable flag in the control register is valid, an interrupt request will be generated after the processing of this descriptor, no matter whether the transmission is correct or not; when it is 0, no interrupt will be generated; bit 14 in place 18 is the error bit, which is set by MAC.

2.3.Related Registers of Ethernet MAC Controller

Table 1 describes the relevant registers of Ethernet MAC controller, which needs to be operated according to the status of MAC control register during Ethernet transmission.

Table 1: Ethernet MAC controller related registers.

Register name	Addr	Function description
MAC control register	0x2000	Bit 6: software reset Bit 3:receive interrupt enable Bit 2:transmit interrupt enable Bit 1: allow to receive Bit 0: allow to transmit
MAC status register	0x2016	Bits5-4:bus communication error Bit3:transmitted an Ethernet packet correctly Bit2:received an Ethernet packet correctly Bits1-0:Ethernet packets transmission or reception error
Transmit descriptor table register	0x206e	Bits31-10:transmit descriptor table base address Bits9-3:transmit descriptor table counter
Receive descriptor table register	0x2084	Bits31-10:receive descriptor table base address Bits9-3: receive descriptor table counter

3. Overall Design of Ethernet Card Driver Based on UM-BUS

3.1.The Architecture of UMBUS Device Driver

According to Figure 1, the device driver of UM-BUS should have two functions: 1. After loading the bus device driver, the master control device in the CPS system can read and write the address space of other bus devices through the UM-BUS access interface, so as to control the sensors and actuators connected by the slave devices; 2. After loading the bus device driver, the master control

device in the CPS system has realized the cross Through the Ethernet interface of UM-BUS, the interaction between ECS and CPS system is realized.

The driver layer of UM-BUS device mainly consists of three modules: PCIe module, UM-BUS module and UM-BUS network card module. The overall design of UM-BUS device driver layer is shown in Figure 3.

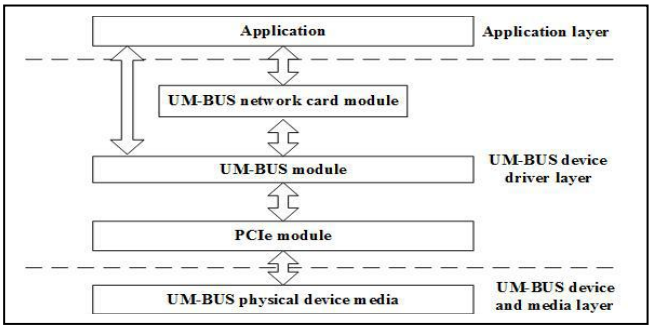

Figure 3: Structure diagram of UM-BUS device driver.

The PCIe module realizes the access to the PCIe device of the bus device media layer, the UM-BUS expansion board connects to the host through the PCIe interface, the access to Ethernet initiated by the upper application program and the access to the UM-BUS bus are finally transmitted to the hardware device through the PCIe module; the UM-BUS module realizes the access to the UM-BUS device, including the storage space, IO space and attributes of the UM-BUS bus read and write access to space; UM-BUS network card module realizes the control of Ethernet card. By loading standard Ethernet driver, the application layer can communicate with other hosts through the network card.

3.2.The Architecture of UM-BUS Network Card Driver

The network system of Linux is mainly based on the socket mechanism of BSD UNIX. There is a special data structure (sk_buff) defined between the system and the driver for data transmission [3]. The operating system supports the cache of transmitting data and receiving data, provides the flow control mechanism, and provides the support of multi protocol [4]. The function of the network card driver of Linux is to transmit the information from the upper layer protocol stack through the network card. The Linux system defines the network device driver as four layers, which are the network protocol interface layer, the network device interface layer, the network driver interface layer and the device media layer [5]. The structure of Linux network device driver is shown in Figure 4.

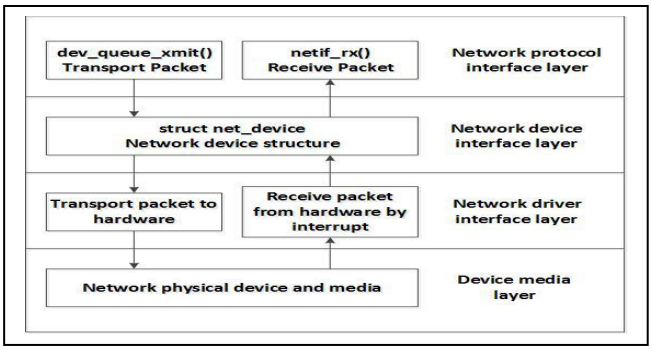

Figure 4: Linux network card driver hierarchy diagram[6]

The network protocol interface layer is a protocol to realize unified packet receiving and transmitting. It provides a unified packet receiving and transmitting interface to the network protocol. This layer is mainly responsible for calling the dev_queue_xmit() function to transmit packets, and the netif_rx() function to receive packets, using sk_buff as the data carrier.

The network device interface layer is the key part of the whole network interface. It describes the information of a specific network device through the net device network device structure, defines the transmitting and receiving functions, realizes the unification of different hardware, and registers the net device into the kernel [7].

The main work of the UM-BUS network card driver is to initialize the network card device and realize the transmitting and receiving of the network card device.

3.3.UM-BUS Device Access

In the UM-BUS controller, the destination node number register, destination address register, access data register, bus communication command register, DMA transmission control register and bus status register are set. The device access of UM-BUS is controlled by operating the relevant registers of UM-BUS controller. The bus access is divided into single word, IO and DMA access of UM-BUS.

Single word and IO access process of UM-BUS: access the relevant registers of UM-BUS controller through PCIe, write the target node number and target address to the corresponding registers. If it is a write operation, it also needs to write the access data, then write the access command, start the access, if it is a read operation, it needs to read the returned data. Through the PCIe access bus status register to determine whether to complete the bus access operation.

DMA access process: initialize the related registers, control the DMA transfer between PC memory and Ethernet data buffer through DMA transfer control register.

3.4.The Implementation of UM-BUS Network Card Driver

3.4.1. Initialization of Network Card Device

The initialization function of the network card device is the first step of the implementation of the network card driver. This function is responsible for initializing the Ethernet controller, allocating and initializing the net device object, setting the contract function [8] in the net device structure member, and registering the network device with the system.

The initialization process of Ethernet controller is as follows: first, initialize MAC control word, perform initialization operation on PHY, including reading PHY identification register, resetting PHY, configuring PHY working mode, starting automatic negotiation, and finally generating MAC control word, then setting MAC address and control word, clearing transmit descriptor buffer and receive descriptor buffer through DMA write operation of bus, and passing through bus Single word write sets MAC control word, initializes Ethernet receive buffer descriptor table, writes all constructed descriptors to receive descriptor buffer on board through bus DMA write operation, and finally starts MAC receive.

3.4.2. Transmission of Network Card Packets

The upper layer passes the packet to the packet function in the driver through sk_buff. The packet function is responsible for transferring the Ethernet packet to the memory of the bus controller through UM-BUS bus, and then transmitting it out through the Ethernet controller.

The transmitting process of Ethernet controller is as follows:

① Read the descriptor control word of the transmit buffer;

② If bit 11 is 1, the transmitting buffer is full and it needs to wait for the current description transmission to complete. Return ①;

③ If bit 11 is 0, it means that the transmit buffer is empty. Write new data and descriptors to the transmit buffer. First write the address word of the buffer. After writing, correct the pointer of the transmit buffer. Then write the control word of the descriptor. After writing the descriptors, start MAC transmission;

④ Read the MAC control word to see if it is being sent. If bit 0 is 0, it means that the MAC is not in the transmitting state at present, then transmit the data written to the buffer this time. Write the address of the descriptor written this time to the register of MAC transmission descriptor table, and then set the MAC control word bit 0 to 1 to start the transmission of the current descriptor block;

⑤ If the MAC control word bit 0 is 1, it means that the MAC is in the process of transmitting at present, and the transmitting packet added this time has been added to the transmitting chain list, it will be sent automatically according to its order in the buffer;

⑥ The transmitting of a packet has been started successfully and exited normally.

The above operations are conducted 8 times of UM-BUS device access.

3.4.3. Receiving and Interrupt Processing of Network Card Packets

The main method of receiving data for network card device is to cause the interrupt service program of the device by the interrupt service program. The interrupt service program judges the interrupt type. If it is the interrupt of receiving Ethernet packet, it will enter the Ethernet controller's packet receiving function to receive the Ethernet packet, and then transfer the Ethernet packet from the receiving buffer to the host memory through DMA. After receiving the Ethernet packet, the host will construct a sk_buff to put the data in and submit it to the upper layer through the netif_rx() function.

Considering the asynchrony of receiving and transmitting, eight receiving buffers are opened by using the hardware buffer to form a circular linked list. When the last descriptor is reached, it needs to be rolled back to the first one, and the pointer sequence number needs to return to 0.

The receiving process of Ethernet controller is as follows:

① Read the descriptor control word of the receive buffer;

② If bit 11 is 0, it means that the buffer corresponding to the descriptor has received a datagram. The data is moved to the user buffer through DMA, a sk_buff is constructed, the data is written into sk_buff and submitted to the upper layer, and descriptor control word correction, re-enable the descriptor for subsequent reception, and generate an interrupt for corresponding processing, modify the buffer pointer sequence number, and reach 0 when it reaches the seventh;

③ If bit 11 is not 0, it means that no packet has been received and the loop is exited;

④ If reception has been stopped, restart reception, read MAC control word,

⑤ If bit 1 is 0, it means that reception has been stopped. To restart reception, write the start address of the receive descriptor to the register of the receive descriptor table, and then set MAC control word bit 1 to 1 to start reception of the current descriptor block. Return ①.

The above operations are conducted for 6 times of UM-BUS device access.

4. Testing of the Driver

4.1.Test Environment

In order to verify the correctness and validity of the drive of UM-BUS network card, the test system consists of a PC machine equipped with UM-BUS expansion card, a UM-BUS expansion card and a standard PC machine, which are connected by UM-BUS. The operating system used in the test environment is Linux.

4.2.Connectivity Test

After connecting successfully according to the organization mode of the test environment, compile the written driver into the kernel and execute it. Execute ifconfig command to check the relevant Ethernet network card information. By executing ping command, both sides can communicate. See Figure 5 [9] for details.

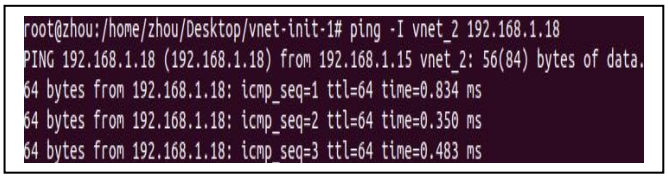

Figure 5: Test result chart.

The third-party software, pigeon transmission, was used for file transmission test. The size of the transmission file was 2G, and the transmission was 100 times, without packet loss. The function of using HTTP to browse the web page was correct.

4.3.Transmission Performance Test

UDP communication is carried out in the test environment of Figure 6, and Wireshark packet capturing tool is used for monitoring. The IP address of the main control device is 192.168.1.18, and the IP address of the target device is 192.168.1.15. See Figure 6 for details.

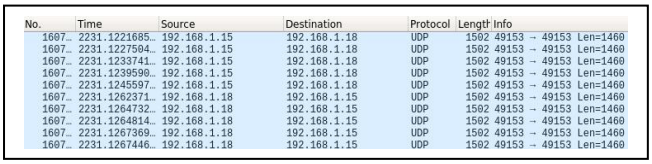

Figure 6: UDP communication test result chart.

UDP packet transmission is shown in Table 2. The transmission file is 10G, and the average transmission rate is 3.787MB/S. There is no packet loss and no data error is found during the transmission.

Table 2: Transmission statistics.

Transfer file size	Average transmission speed	Packet loss	Error
10G	3.787MB/S	0	0

Acknowledgments

This work was financially supported by National Natural Science Foundation of China (61772350); pre research project of shared information system equipment (open) (No.JZX2017-0988 / Y300);

Beijing Science and Technology New Star Program (Z181100006218093); open project of National Key Laboratory of Architecture (CARCH201607); research fund support project of Beijing future chip technology advanced innovation center (KYJJ2018008); science and technology innovation service ability Power construction - basic scientific research business cost (Scientific Research) (006-19530050173).

References

[1] Weigong Zhang, Jiqin Zhou,Jie Li,et al,UM-BUS and a Plug and Play Architecture,Acta Electronica Sinica, 2015, vol.43:9, pp.1776-1785.

[2] Renfa Li, Yong Xie,Rui Li, et al,Survey of Cyber-Physical Systems, Journal of Computer Research and Development, 2012, vol.49:6.

[3] Nannan Zhao, Jingwei Zhao,Keyan Zhul, DM9000 Network Driver Based on Embedded Linux,Journal of University of Science and Technology Liaoning, 2014, vol.37:3, pp.265-268.

[4] Anfu Yuan,Shengfeng Xia,Design and Implementation of the DM9000 Network Device Driver Based on ARM and Linux, Computer Engineering&science, 2011, vol.33:2, pp.27-31.

[5] Feng Chang, Chuangliang Meng, et al,Development of Network Driver Program Based on Embedded Linux,Communications Technology, 2012, vol.45:6, 2012, 36-39.

[6] Xinhui Long, Junjie Chen, Design and Implementation of Ethernet Driver Based on Embedded Linux, 2011, vol.31:3, pp.149-152.

[7] Hongwei Xie, Chunyan Song, The Research and Implementation of Network Card Drivers of Embedded ARM-Linux, Computer Development & Applications, 2012, vol.25:4, pp.54-57.

[8] Hong Zhang, Qinzhang Wu, Chunlei Du, System Design of Testing Platform Based on Linux Virtual Network,Electronic Design Engineering, 2016, vol.24:17, pp.96-97.

[9] Song Gao, Chao Ji, Chaobo Chen, DM9000 Network Driver Design Based on Embedded Linux, Computer&Digital Engineering, 2013, vol.41:2, pp.156-158.