Design and implementation of the LwIP in embedded system

Man-Qing Hu†, Geng-Xin Liu, Xiao-He Liu and Yi-Fei Jian,
*Chengdu College of University of Electronic Science and Technology of China,
Chengdu 611731,China
E-mail: 30742144@qq.com
www.cduestc.cn*

This paper realizes TCP/IP communication basing on EMAC of TMS320DM642 as the design platform. This paper analyzes the operation principle of the modules and the structure of light-weight TCP/IP protocol stack. Through the transplantation of LwIP in TMS320DM642, we realize the basic functionality of the network and make it work stably, providing a solid solution for TM320DM642 to use in embedded system.

Keywords: TMS320DM642; EMAC; TCP/IP protocol stack; transplantation.

1. Introduction

As a solution to realize communication through Internet, TCP/IP is a hot spot in embedded system research. LwIP is a wide-used open source light-weight TCP/IP protocol, which not only maintain the main functionality of TCP protocol, but also reduce the occupation of RAM.

TMS320DM642 is a multimedia specified DSP chip with clock rate as high as 600 MHz, rich peripheral interface circuit like audio and network communication. It comprises modules such as VP video communication, HPI interface, EMAC network communication. For the sake of secondary development, this paper chooses UC/OS-II and LwIP to realize network module development.

2. EMAC in TMS320DM642

EMAC in TMS320DM642 comprises 3 parts: EMAC control module, EMAC transmitting module and MDIO module.

EMAC control module is the main channel of communication among DSP Kernel, EMAC transmitting module and MDIO module. It controls the priority of reboot, interrupt and storage. With 4K Bits memory, it can save 256 Internet data packages without the interference of DSP.

MDIO module can monitor the status of network interface chip BCM5221 and configure connection parameters of the chip. With this module, every network that connects to the chip can receive and transmit data according the request.

EMAC transmitting module is the interface between DSP process and external network. Its receive part comprises DAM engine, receive FIFO an MAC receiver. And the transmitting part comprises DAM engine, transmitting FIFO an MAC transmitter. The control system of EMAC transmitting module comprises control register and SRAM.

Within the whole EMAC module, data transmitting and receiving is realized by data package buffer file descriptor. At EMAC initializing, data package buffer file descriptors for transmitting and receiving is set up respectively. DAP can automatically read these descriptors and hence reading or writing data accordingly. When set up the descriptors, one must follow the format, or DM642 cannot automatically read these descriptors. The descriptor is shown in Tab.1.

Tab. 1. Descriptor of a data package buffer file

Word offset	Bit fields	
	16	15
0	Next descriptor pointer	
1	Buffer pointer	
2	Buffer offset	Buffer length
3	Flags	Packet length

Pointer points to the address of next descriptor where next descriptor. Buffer pointer points to the address of current data. Buffer offset is the offset of current buffer compare to the address pointed by Buffer pointer. Buffer length is the length of current buffer descriptor. Flags means whether the buffer descriptor is the starting or ending of the data package. Package length is the length of the data package.

All communication of bottom level data of EMAC is performed by TI's on-chip supporting library. Before initializing EMAC, the signal data buffer descriptor need to be set up according to EMAC_Pkt structure. Also, sufficient space need to be allocated in pDataBuffer as the data buffer for receiving and transmitting. The program is shown as follows:

```
Packet[i].pDataBuffer = RxBuffer;
Packet[i].BufferLen =1500;
Packet[i].ValidLen =1500;
Packet[i].DataOffset =0;
Packet[i].PktChannel =0;
```

```
Packet[i].PktLength =1500;
Packet[i].PktFrags =1;
```

Next, based on EMAC_Config, we need to configure the EMAC module, especially for the two callback function, EMAC_Pkt*(*pfcbGetPacket) and EMAC_Pkt*(* pfcbRxPacket), who will allocate and retrieve descriptor during the initializing. The code for EMAC_Config is shown as follows:

```
typedef struct_EMAC_Config{
unit   ModeFlags;
unit   MdioModeFlags;
unit   TxChannels;
unit8   MacAddr
unit   RxMaxPktPool;
EMAC_Pkt*(*pfcbGetPacket)(Handle   hApplication);
Void   (*pfcbFreePacket)(Handle   hApplication, EMAC_Pkt*pPacket);
EMAC_        Pkt*(*pfcbRxPacket)(Handle                      hApplication,
EMAC_Pkt*pPacket);
Void (pfcbStatistics)( Handle   hApplication);
}EMAC_Config;
```

Finally, we need to call API function EMAC_open( ), thus the system will automatically setup receiving and transmitting descriptors, build structure of EMAC_DesCh, which is shown as follows:

```
typedef struct_EMAC_DescCh{
   stuct_EMAC_Device*pd;
PKTQ       DescQueue;
PKTQ       WaitQueue;
unit        ChannelIndex;
unit        DescMax;
unit        DescCount;
EMAC_Desc     *pDescFirst;
EMAC_Desc     *pDescLast;
EMAC_Desc     *pDescRead;
EMAC_Deac     *pDescWrite;
}EMAC_DescCh;
```

During the process of system receiving, system will use pointers pDescWrite and pDescRead to point out the data need to receive or transmit. According to the sequence of the buffer descriptor, DSP will save the data to address space of pDataBuffer and constantly move pointer. When an interrupt is produces by data transfer, EMAC_severceCheck( ) can be called to receive the data. When the pointer of pDescRead is unequal to that of pDescWrite, the pointer will keep moving forward from Read until they are equal. Every pointer moving will call the callback function pfcbRxPacket to save the data. When data transmitting, it will include the data into the link DescQueue, and transmit the data by moving DescWrite in transmitting buffer descriptor.

3. Transplantation of LwIP Protocol Stack

3.1. *Introduction*

LwIP is an open source light weight IP protocol stack which can be transplanted into all kinds of operating systems and can operate without an operating system. LwIP protocol stack includes 3 levels. The first level is operating system simulation level, on which operators of LwIP have already written a unified interface, and users can direct input program to the interface to implement functionality. The second level is network interface level, on which users connect network and bottom level drive according to situation of bottom level drive. The third level is protocol level, which analyzes and combines the data package. The implementation of LwIP main loop is as follows. The main loop task is set when LwIP is initializing. Next, it will call function sys_mbox_fetch() and use mailbox to block the task. When there is input data, IP data package is inputted using ip_input(). After analyzing the package header in protocol stack, according mailbox signal is sent based on the information in package header. Thus the main loop task will rework and step into according analyzing program.

3.2. *Initialization of LwIP*

The whole LwIP protocol stack is realized based on UC/OS-II operating system. This section LwIP will connect LwIP and EMAC module of TMS320DM642.

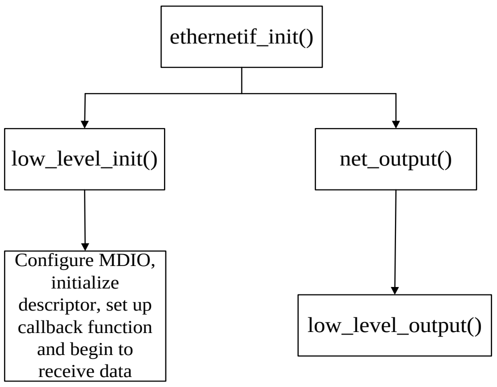
ethernetif_init()
low_level_init()
net_output()
Configure MDIO, initialize descriptor, set up callback function and begin to receive data
low_level_output()

Fig. 1. Initialization of LwIP

Fig. 1 shows the initialization of LwIP. In the initialization main function ethernetif_init(), to get ready for transmitting data package, callback function net_output and low_level_output are assigned with netif->output and netif-

>linkoutput, respectively. Next, EMAC is initialized through low_level_init(). In low_level_init(), system need to complete the basic configuration of EMAC module. Through MDIO module, existing PHY is searched and initialized. Based on what is described in section 2, EMAC is initialized and according callback function is set. After the initialization, one can use EMAC module to transfer data.

3.3. *Receive data package of LwIP*

The receiving process of LwIP is shown in Fig. 2. First of all, one need to open the according EMAC interrupt in DM642 so that when data package is input to EMAC, program will generate the interrupt. Then call TI's API function EMAC_serviceCheck() in the interrupt process to receive according data package. The most important function in the process is callback function pfcbRxPacket. Because it need to transmit the data received to package buffer of LwIP and then retrieve receive descriptor to get ready for the next receive. Also, the program calls ethernetif_input() in LwIP to accomplish the according job. First, low_level_input( ) in ethernetif_input() is called to open LwIP package buffer that matches the package. Then the data in receive descriptor is copied to package buffer of LwIP and the package header is analyzed using htons(). If it is a IP package header, ctharp_ip_input() and netif->input() will be used to enter analyze program in LwIP. If it is a ARP package header, etharp_arp_input() will be used to complete APR analyze and response.

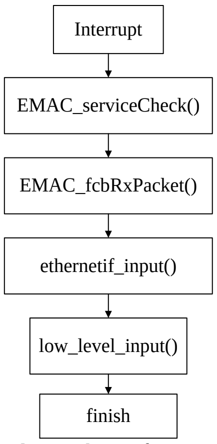

Fig. 2. Receive data package of LwIP

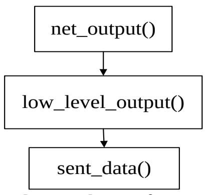

Fig. 3. Transmitting data package of LwIP

3.4. *Transmitting data package of LwIP*

The process of transmitting LwIP data is shown in Fig.3. At initialization, the program has already set netif->output and netif->linkoutput. The callback function low_level_output()of netif->linkoutput firstly copy the data in package buffer of LwIP to a temporary space. Then send_data() is called to transmit data packages. Function send_data() set buffer descriptor according to the structure of it then send the address of the temporary space to pDataBuffer. Finally, EMAC_sendPacket() is called to transmit the data.

4. Summary

In the process of transplanting LwIP to TMS320DM642, one need to set ETH_PAD_SIZE in user setting file to 2. Because the compiler of TMS320DM642 require that Bits must align, and it will cause misalign without this setup. After experiment, the transplanted LwIP can fulfill basic network function and work stably.

Acknowledgments

The authors would like to thank the Funds of Sichuan Provincial Department of Education (No. 14ZB0386) for their support and assistance.

References

1. Zheng Xiangqian, Yuan Yue. Design and Imple mentation of embedded network based on LwIP[J]. Joural of Measurement Science and Instrumentation, 2010, 16(1):30-33.
2. Dunkels A. Design an implementation of the lwIP TCP/IP Stack[R]. Stockholm: Swedish Institute of Computer Science, 2001.
3. Magdalena Iovescu, Mike Denio. Software Operation of Gigabit Ethernet Media Access Controller on TMS320C645x DSP[Z]. Texas Instruments, 2006.
4. Zhang Ying. The Research an Implentantion of Embedded Modbus/TCP protol[D]. Hangzhou: Zhejiang University. 2008: 20-22.
5. Huang Linsheng, Lin Yan. Realization of Embedded TCP/IP Stack on μC/OS-II Based on DSP[J]. Computer Technology and Development, 2008, 06.
6. B Ravat, A Diwan and PSK Reddy. Development of an Arm Based Modbus RTU to TCP/IP Protocol Converter using μC/OS II[J], Networking & Communication Engineering, 2013, 5(6).