Application Note

Application Note

Document No.: AN1088

APM32F4xx Series ETH Transplantation LWIP Application Note

Version: V1.0

1 Introduction

This application note provides a guide to how to configure and apply ETH peripherals on EVAL Board of APM32F4xx series, including transplantation of lwip protocol stack, code implementation and application methods.

APM32F4xx MCU provides configurable and flexible Ethernet peripherals to meet various application requirements of customers. It supports two industry standard interfaces connected to the external physical layer: MII and RMI used by default, and MII is only defined in IEEE 802.3 specification. It has many application fields, such as switch, network interface card and so on. Ethernet can transmit and receive data according to IEEE 802.3-2002 standard with the help of peripherals.

Ethernet complies with the following standards:

IEEE802.3-2002 for Ethernet MAC

IEEE1588-2008 standards which specify synchronization precision of networked clock

AMBA 2.0 for AHB master/slave ports

RMII specification of RMII Alliance

Contents

2 Introduction to Development Environment

This chapter mainly introduces Ethernet, ETH peripherals of APM32F407 and EVAL board related hardware design.

2.1 Introduction to Ethernet

Ethernet is a kind of computer LAN technology, which complies with IEEE 802.3 standard. IEEE 802.3 specifies the content of connection, electronic signal and media access layer protocol of physical layer. It has many application fields, such as switch, network interface card and so on.

2.1.1 Physical layer

In the physical layer, the IEEE 802.3 standard specifies the transmission media, transmission speed, data coding method and conflict detection mechanism used by Ethernet. In the actual development process, the physical layer generally implements the function of the physical layer through a PHY chip. The development board we use in this application is APM32F407 EVAL board, and the on-board PHY chip is LAN8720A, which is a small, low-power Ethernet physical layer transceiver, and only supports RMII interface.

2.1.2 MAC sublayer

APM32F407/417xExG series product internally integrates media access control (MAC 802.3), completes the functions of MAC sublayer, and is responsible for data handover with the physical layer.

2.1.3 TCP/IP protocol stack

TCP/IP is a data communication mechanism. The implementation of the protocol stack is essentially to process data packets. LwIP data packet management provides an efficient processing mechanism.

2.1.3.1 Introduction to LwIP

LwIP is a lightweight TCP/IP protocol, and a small open-source TCP/IP protocol stack developed by AdamDunkels of the Swedish Institute of Computer Science (SICS). The occupation of resources is reduced while the main functions of TCP protocol are maintained. In addition, LwIP can be ported to the operating system and run independently without an operating system. The source code packet of each version of LwIP and the corresponding contrib packet can be downloaded and obtained on the web page http://savannah.nongnu.org/projects/lwip/. The version used in the routine is lwip-1.4.1.

2.1.3.2 Main features of LwIP

1. Support ARP protocol (Ethernet Address Resolution Protocol).

2. Support ICMP Protocol (Internet Control Message Protocol) for network debugging and maintenance.

3. Support IGMP protocol (Internet Group Management Protocol), which can implement receiving of multicast data.

4. Support UDP protocol (User Datagram Protocol).

5. Support TCP protocol (Transmission Control Protocol), including congestion control, RTT estimation, fast recovery and fast forwarding.

6. Support PPP protocol (Point-to-point Protocol) and PPPoE.

7. Support DNS (Domain Name Resolution).

8. Support DHCP protocol and dynamically allocate IP addresses.

9. Support IP protocols, including IPv4 and IPv6 protocols, support IP fragmentation and reassembly functions, and forwarding of data packets under multiple network interfaces.

10. Support SNMP protocol (Simple Network Management Protocol).

11. Support AUTOIP and automatic IP address configuration.

12. Provide a special internal callback interface (Raw API) to improve application program performance.

13. Provide optional Socket API and NETCONN API (used in multithreaded scenarios).

2.1.3.3 Programming interface of LwIP

LwIP provides three programming interfaces, namely, RAW/Callback API, NETCONN API and SOCKET API. SOCKET API has the highest usability and the lowest execution efficiency, while RAW/Callback API has the lowest usability and the highest execution efficiency. Users need to choose according to the actual situation.

2.2 ETH peripherals of APM32

Ethernet peripherals include MAC 802.3 with dedicated DMA controller. It supports MII and RMII used by default and switches them through select bit. It also includes SMI for communicating with external PHY. The mode and function of MAC controller and DMA controller can be selected through a configuration register.

When transmitting data, first transmit the data from the system memory to the TX FIFO buffer through DMA, and then transmit it through the MAC core. The Ethernet frame received through the line is stored by RX FIFO before it is transmitted to the system memory through DMA.

2.2.1 Station management interface (SMI)

SMI supports accessing 32 PHY. The application program selects one PHY among the 32 PHY through the 2-wire clock and data line, and then accesses any PHY register. Only one register in one PHY can be addressed at any given time.

SMI write operation: When MB bit and MW bit of MAC_ADDR are set to 1 by application program, SMI will trigger write operation of PHY register by transmitting PHY address, and register address in PHY, and writing data. When performing write operation, the application program cannot modify MAC_ADDR and MAC_DATA registers. After write operation is completed, SMI will reset MB bit.

SMI read operation: When MB bit of MAC_ADDR is set and MW bit is cleared to zero, SMI will trigger read operation of PHY register by transmitting PHY address, and register address in PHY. When performing read operation, the application program cannot modify MAC_ADDR and MAC_DATA registers. After read operation is completed, SMI will reset MB bit, and update the read data from PHY to MAC_DATA register.

2.2.2 Media independent interface (MII)

MII defines the interconnection between MAC sublayer and PHY at data transmission rate of 10Mbit/s and 100Mbit/s. The signals are as follows:

MII_TX_EN: Transmit enable signal; MAC is currently transmitting half byte for MII

MII_RX_DV: Data receiving effective signal; PHY is currently receiving recovered and decoded half byte of MII

MII_TXD [3:0]: Data transmitting signal

MII_RXD [3:0]: Data receiving signal

MII_RX_ER: Receiving error signal

MII_TX_CLK: Continuous clock signal, providing reference timing for TX data transmission

MII_RX_CLK: Continuous clock signal, providing reference timing for RX data transmission

MII_CRS: Carrier sense signal

MII_COL: Conflict detection signal

MII clock source: 25MHz clock must be provided to external PHY to generate TX_CLK and RX_CLK clock signal; this signal is output through MCO pin. The required frequency can be obtained on MCO pin through 25 MHz external quartz crystal only when PLL frequency multiplier is configured.

2.2.3 Reduced media independent interface (RMII)

RMII reduces the number of pins of MCU of Ethernet peripherals and external PHY at 10/100Mbit/s. According to IEEE 802.3u standard, MII has 16 data and control signal pins. RMII reduces the number of pins to 7.

RMII is instantiated between MAC and PHY. It is conductive to converting MII of MAC to RMII. RMII has the following characteristics:

Separate 2-bit wide transmitting and receiving data path

10-Mbit/s and 100-Mbit/s running speed

The reference clock is 50MHz

Provide the same reference clock to MAC and external Ethernet PHY from the outside

RMII clock source: Use external 50MHz clock or embedded PLL to generate 50MHz-frequency signal to drive PHY.

2.3 Hardware design

The development board uses APM32F407 controller to connect to LAN8720A Ethernet PHY through RMII interface and SMI interface.

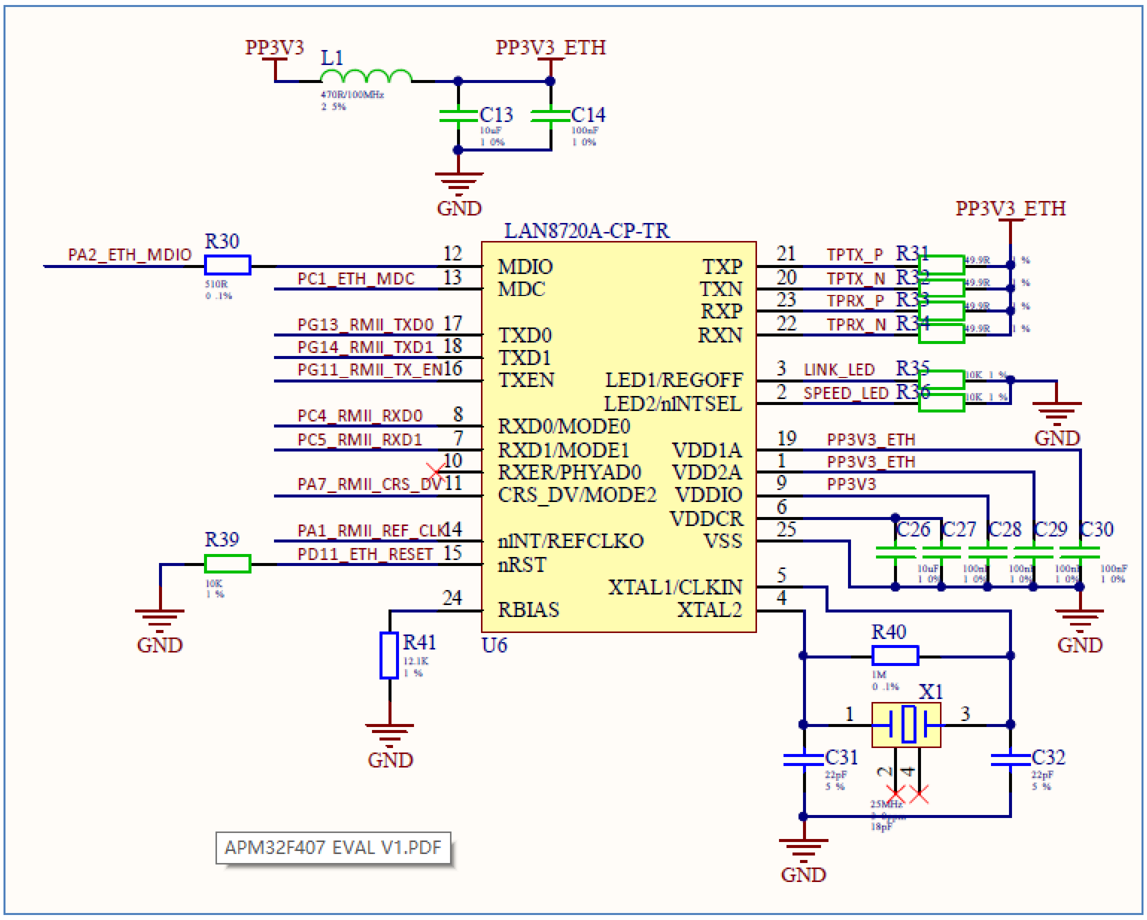


Figure 1 LAN8720A Hardware Circuit

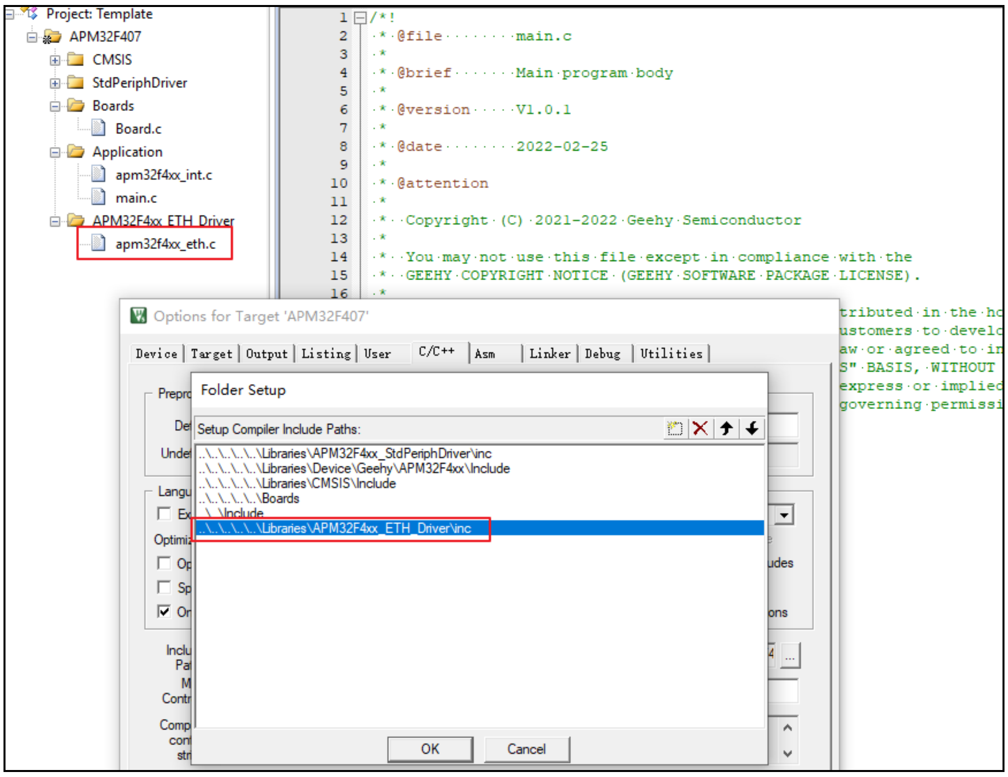
Set the nINTSEL pin to low level by connecting the pull-down resistor, to enable the output function of nINT/REFCLKO pin to provide clock signal for REF_CLK signal line in RMII interface; connect between XTAL1 and XTAL2 on the hardware to provide 25MHz clock; after the frequency doubling of LAN8720A internal PLL circuit, the output clock signal of nINT/REFCLKO pin is 50MHz clock.

3 Transplant LwIP

This chapter mainly introduces how to transplant LwIP to APM32F407 bare metal project without operating system, and shows the experimental demonstration of ping instruction.

3.1 PHY underlying driver

Because there are no files related to ETH peripheral in the standard library, you need to first add the ETH driver library files apm32f4xx_eth.c and apm32f4xx_eth.h in the bare metal project to implement ETH drive.



```
/*!
 * @file        main.c
 *
 * @brief       Main program body
 *
 * @version     V1.0.1
 *
 * @date        2022-02-25
 *
 * @attention
 *
 *  Copyright (C) 2021-2022 Geehy Semiconductor
 *
 *  You may not use this file except in compliance with the
 *  GEEHY COPYRIGHT NOTICE (GEEHY SOFTWARE PACKAGE LICENSE).
 *
```

```
..\..\..\..\..\Libraries\APM32F4xx_StdPeriphDriver\inc
..\..\..\..\..\Libraries\Device\Geehy\APM32F4xx\Include
..\..\..\..\..\Libraries\CMSIS\Include
..\..\..\..\..\Boards
..\..\Include
..\..\..\..\..\Libraries\APM32F4xx_ETH_Driver\inc
```

Figure 2 Add Driver Library

Create new underlying driver files board_LAN8720A.c and board_LAN8720A.h to realize the PHY related drive, initialize the GPIO of RMII interface used by Ethernet, and complete the configuration of MAC and DMA of ETH. Finally, call ConfigEthernet( ) in the main function to drive the network card.

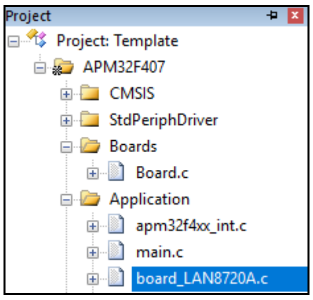
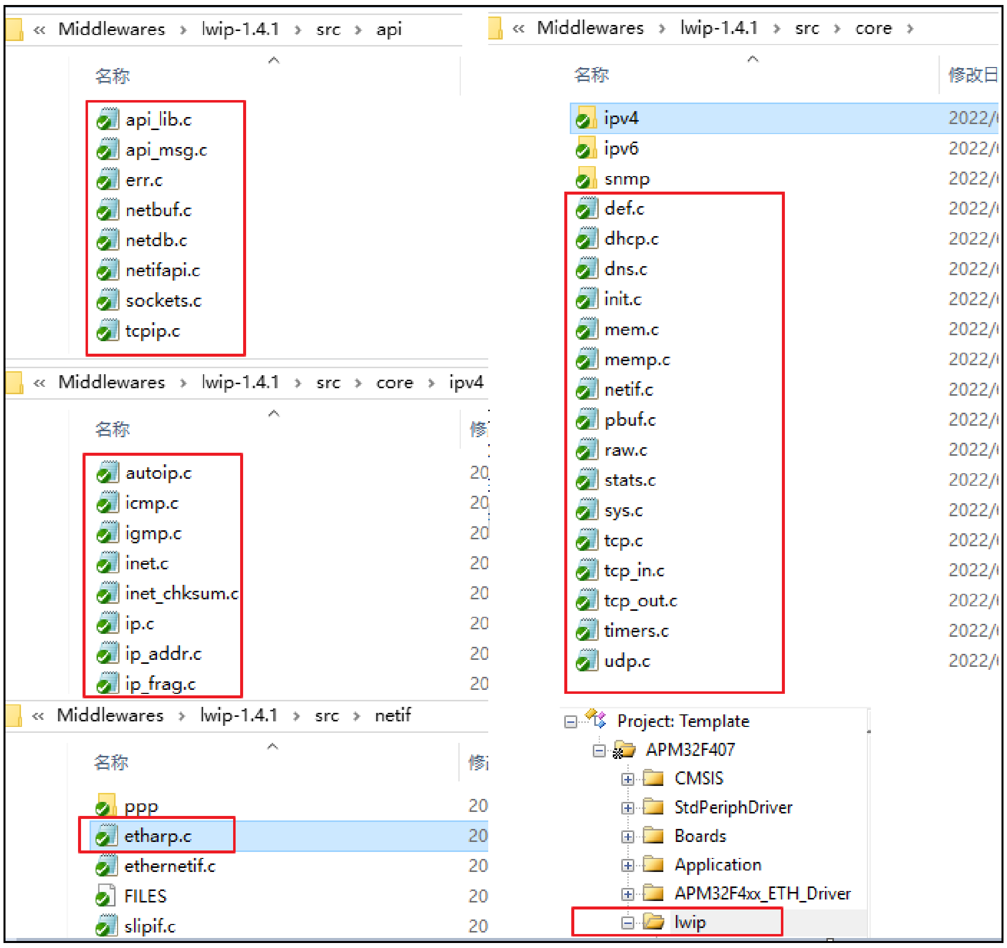


Figure3 Create New Underlying Driver Files

3.2 Add LwIP source file

In the project with Ethernet PHY driver written, create a new group lwip and add the following files into the project without modification.


« Middlewares › lwip-1.4.1 › src › api

名称

api_lib.c
api_msg.c
err.c
netbuf.c
netdb.c
netifapi.c
sockets.c
tcpip.c

« Middlewares › lwip-1.4.1 › src › core › ipv4

名称 修

autoip.c
icmp.c
igmp.c
inet.c
inet_chksum.c
ip.c
ip_addr.c
ip_frag.c

« Middlewares › lwip-1.4.1 › src › netif

名称 修

ppp
etharp.c
ethernetif.c
FILES
slipif.c

« Middlewares › lwip-1.4.1 › src › core ›

名称 修改日

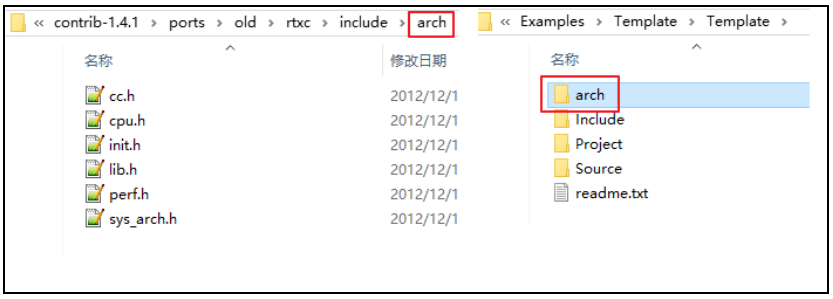
ipv4
ipv6
snmp
def.c
dhcp.c
dns.c
init.c
mem.c
memp.c
netif.c
pbuf.c
raw.c
stats.c
sys.c
tcp.c
tcp_in.c
tcp_out.c
timers.c
udp.c

Project: Template
APM32F407
CMSIS
StdPeriphDriver
Boards
Application
APM32F4xx_ETH_Driver
lwip

Figure 4 LWIP Source Files

3.3 Transplant header files

After LWIP source files are transplanted, our project also needs some corresponding header files to support copying the rectification arch folder in contrib-1.4.1\ports\old\rtxc\include\arch downloaded from LWIP website to the current project and contain it.


« contrib-1.4.1 › ports › old › rtxc › include › arch

名称	修改日期
cc.h	2012/12/1
cpu.h	2012/12/1
init.h	2012/12/1
lib.h	2012/12/1
perf.h	2012/12/1
sys_arch.h	2012/12/1

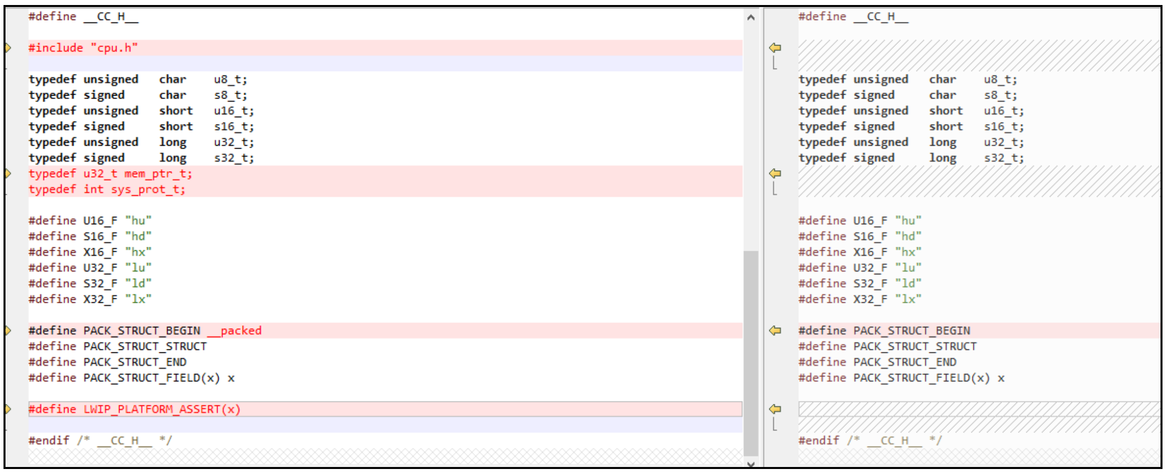
« Examples › Template › Template ›

名称

arch
Include
Project
Source
readme.txt

Figure 5 LWIP Header Files

Because we use Keil development tools, we need to simply modify cc.h file. The modified file is shown below on the left.

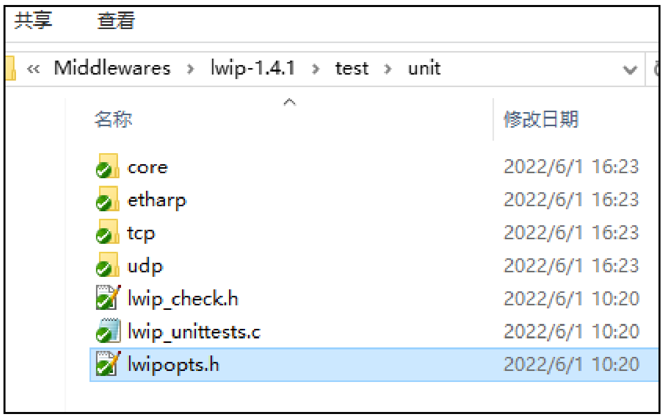


```
#define __CC_H__

#include "cpu.h"

typedef unsigned   char    u8_t;
typedef signed     char    s8_t;
typedef unsigned   short   u16_t;
typedef signed     short   s16_t;
typedef unsigned   long    u32_t;
typedef signed     long    s32_t;
typedef u32_t mem_ptr_t;
typedef int sys_prot_t;

#define U16_F "hu"
#define S16_F "hd"
#define X16_F "hx"
#define U32_F "lu"
#define S32_F "ld"
#define X32_F "lx"

#define PACK_STRUCT_BEGIN __packed
#define PACK_STRUCT_STRUCT
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x

#define LWIP_PLATFORM_ASSERT(x)

#endif /* __CC_H__ */
```

```
#define __CC_H__

typedef unsigned   char    u8_t;
typedef signed     char    s8_t;
typedef unsigned   short   u16_t;
typedef signed     short   s16_t;
typedef unsigned   long    u32_t;
typedef signed     long    s32_t;

#define U16_F "hu"
#define S16_F "hd"
#define X16_F "hx"
#define U32_F "lu"
#define S32_F "ld"
#define X32_F "lx"

#define PACK_STRUCT_BEGIN
#define PACK_STRUCT_STRUCT
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x

#endif /* __CC_H__ */
```

Figure 6 cc.h File

cc.h file contains processor related variable types, data structures, and related macros of byte alignment. A series of names such as U16_F, S16_F, X16_F are used for debugging information output format of the debugging function of LwIP.



名称	修改日期
core	2022/6/1 16:23
etharp	2022/6/1 16:23
tcp	2022/6/1 16:23
udp	2022/6/1 16:23
lwip_check.h	2022/6/1 10:20
lwip_unittests.c	2022/6/1 10:20
lwipopts.h	2022/6/1 10:20

Figure7 Add lwipopts.h File

Next, you need to configure the LwIP function options, and copy the file lwipopts.h in the path of lwip-1.4.1\test\unit\ to the project for modification. lwipopts.h header file can clip the LwIP functions, such as whether there is an operating system, memory space allocation, storage pool allocation, TCP function, UDP function selection, programming interface enable, etc. If the user does not configure in lwipopts.h file, LwIP will use default parameters of opt.h.

3.4 Transplant network card driver files

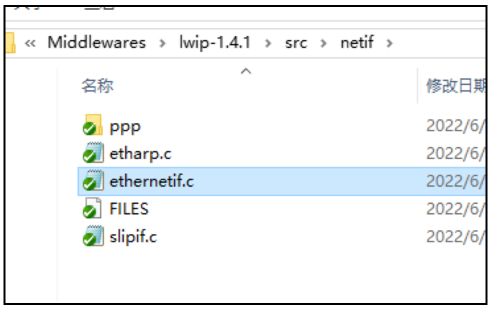


Figure 8 Transplant Network Card Driver Files

ethernetif.c file in lwip-1.4.1 is the template of the driver of the underlying interface, which stores the LwIP and ETH peripheral network interface functions, and will be called when the network card receives or transmits data. We copy this module into the project and modify it according to the implementation method of the network card used. Then the transplantation of LWIP is basically completed.

The main functions that need to be implemented by users are low_level_init, low_level_output and low_level_input, which are used to initialize and enable MAC and DMA, transmit data in physical layer and receive data in physical layer respectively.

3.5 Initialize protocol stack

```
void LwIP_Init(void)
{
struct ip_addr ipaddr;
struct ip_addr netmask;
    struct ip_addr gw;
    /** Initializes the dynamic memory heap */
    mem_init();
    /** Initializes the memory pools */
    memp_init();
    IP4_ADDR(&ipaddr, 192, 168, 73, 22);
    IP4_ADDR(&netmask, 255, 255 , 255, 0);
    IP4_ADDR(&gw, 192, 168, 73, 1);
    /** Config MAC Address */
    ETH_ConfigMACAddress(ETH_MAC_ADDRESS0, SetMACaddr);
    /** Add a network interface to the list of lwIP netifs */
    netif_add(&UserNetif, &ipaddr, &netmask, &gw, NULL, &ethernetif_init, &ethernet_input);
    /** Registers the default network interface */
    netif_set_default(&UserNetif);
    /** When the netif is fully configured this function must be called */
    netif_set_up(&UserNetif);
```

The files that need to be added and modified are ready. Before using LWIP protocol stack, LWIP needs to be initialized first.

First complete the initialization of memory management, and complete the initialization of memory heap and memory pool through mem_init and memp_init. Then initialize the variables ipaddr, netmask and gw through IP4_ADDR, set them to the local IP address, subnet mask and gateway address, and then use netif_add to add the Ethernet device, and transfer the IP address, subnet mask, gateway, and addresses of network card device initialization function and Ethernet frame receiving function into the UserNetif variable to complete the registration of the network card. Set the network card UserNetif as the default network communication device through the function netif_set_default. Finally, call netif_set_up to start the network card to complete the initialization of LWIP. It should be noted that the IP address must be set in the same gateway as the router. For example, the IP address of my computer is 192.168.73.122. I need to set the gateway to 192.168.73.1, the mask to 255.255.255.0, and the IP address to 192.168.73 xx.

3.6 Configure LwIP time base

After the initialization of the whole LWIP core, if you want the protocol stack to run normally, you also need to configure a time base for LwIP so that the core can handle various timing events, such as TCP timing and ARP timing tasks;

SysTick is used as the time base timer of LwIP. An interrupt is triggered every 1ms, and the global variable ETHTimer is added by 1. LwIP can judge whether there is a timeout through the time obtained twice, so as to handle corresponding events.

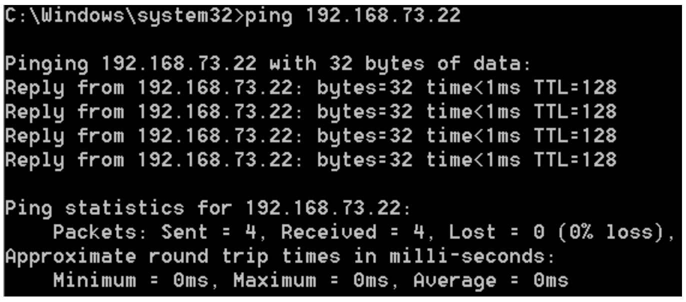
At the same time, a function is also defined in LwIP to realize the timing event sys_check_timeouts(). If you need to use sys_check_timeouts() function, in lwipopts.h header file, set NO_SYS_NO_TIMERS to 0 to support sys_timeout function, implement the function sys_now(), return ETHTimer in the function to obtain the current tick value.

3.7 Acquisition of data packets

At this point, you can use the development board to obtain the network data packets. Generally, you can use polling or interrupt processing to obtain data packets. The following gives an example of obtaining data packets by polling in the current project, and shows the experimental phenomena of ping instruction. By polling, you only need to use ETH_ReadRxPacketSiz periodically in the main function to determine whether a data packet has arrived, and call the ethernetif_input receive function when receiving a data packet.

3.8 Experimental phenomena

After compiling the project and downloading it to the development board, connect the development board and the computer through the network cable, execute ipconfig in the command line through the CMD console of the computer, confirm that the IP address gateway of the computer is 192.168.73.1, and execute ping 192.168.73.22. When you see the experimental phenomena as shown in the figure below, it can be determined that our transplantation work is over, and the network card driver and protocol stack can work normally.



```
C:\Windows\system32>ping 192.168.73.22

Pinging 192.168.73.22 with 32 bytes of data:
Reply from 192.168.73.22: bytes=32 time<1ms TTL=128
Reply from 192.168.73.22: bytes=32 time<1ms TTL=128
Reply from 192.168.73.22: bytes=32 time<1ms TTL=128
Reply from 192.168.73.22: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.73.22:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Figure 9 Experimental Phenomena

4 Revision History

Table 1 Document Revision History

Date	Version	Change History
June 23, 2022	1.0	New

Statement

guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Compliance requirements

Users shall abide by all applicable local laws and regulations when using this manual and the matching Geehy products. Users shall understand that the products may be restricted by the export, re-export or other laws of the countries of the product suppliers, Geehy, Geehy distributors and users. Users (on behalf of itself, subsidiaries and affiliated enterprises) shall agree and promise to abide by all applicable laws and regulations on the export and re-export of Geehy products and/or technologies and direct products.

6. Disclaimer

This manual is provided by Geehy "as is". To the extent permitted by applicable laws, Geehy does not provide any form of express or implied warranty, including without limitation the warranty of product merchantability and applicability of specific purposes.

Geehy will bear no responsibility for any disputes arising from the subsequent design and use of Geehy products by users.

7. Limitation of liability

In any case, unless required by applicable laws or agreed in writing, Geehy and/or any third party providing this manual "as is" shall not be liable for damages, including any general damages, special direct, indirect or collateral damages arising from the use or no use of the information in this manual (including without limitation data loss or inaccuracy, or losses suffered by users or third parties).

8. Scope of application

The information in this manual replaces the information provided in all previous versions of the manual.