应用笔记

Application Note

文档编号：**AN1088**

APM32F4xx 系列 ETH 移植 LWIP

应用笔记

版本：**V1.0**

1 引言

本应用笔记提供如何在 APM32F4xx 系列的 EVAL Board 上配置和应用 ETH 外设的指南，包括 lwip 协议栈的移植、代码实现和应用方法。

APM32F4xx 微控制器提供了可配置、灵活的以太网外设，用以满足客户的各种应用需求。它支持与外部物理层相连的两个工业标准接口：默认情况下使用的 MII 和 RMI，其中 MII 仅在 IEEE 802.3 规范中定义。它有多种应用领域，如交换机、网络接口卡等。且以太网可以借助外设按照 IEEE 802.3-2002 标准发送和接收数据。

以太网遵守以下标准：

用于以太网 MAC 的 IEEE802.3-2002

用于规定联网时钟同步精度的 IEEE1588-2008 标准

用于 AHB 主/从端口的 AMBA 2.0

RMII 联盟的 RMII 规范

目录

2 开发环境介绍

本章主要介绍以太网、APM32F407 的 ETH 外设以及 EVAL 板相关硬件设计。

2.1 以太网简介

以太网是一种计算机局域网技术，遵守 IEEE 802.3 标准，IEEE 802.3 规定了包括物理层的连线、电子信号和介质访问层协议的内容。它有多种应用领域，如交换机、网络接口卡等。

2.1.1 物理层

在物理层，由 IEEE 802.3 标准规定了以太网使用的传输介质、传输速度、数据编码方式和冲突检测机制，在实际开发过程中物理层一般是通过一个 PHY 芯片来实现物理层的功能，在本次应用中我们所使用的开发板 APM32F407 EVAL 板，板载的 PHY 芯片使用的是 LAN8720A，这是一款体积小、功耗低、的以太网物理层收发器，接口方式仅支持 RMII 接口。

2.1.2 MAC 子层

APM32F407/417xExG 系列产品内部集成了介质访问控制（MAC 802.3），完成 MAC 子层功能，负责与物理层进行数据交接。

2.1.3 TCP/IP 协议栈

TCP/IP 是一种数据通信机制，协议栈的实现本质上就是对数据包进行处理，LwIP 数据包管理提供了一种高效处理的机制。

2.1.3.1 LwIP 的简介

LwIP 是一款轻量化的 TCP/IP 协议，是瑞典计算机科学院(SICS)的 AdamDunkels 开发的一个小型开源的 TCP/IP 协议栈。在保持 TCP 协议主要功能的基础上减少对资源的占用。此外 LwIP 既可以移植到操作系统上运行，也可以在无操作系统的情况下独立运行。可以在网页 http://savannah.nongnu.org/projects/lwip/，下载获取到 LwIP 的各个版本的源代码包和对应的 contrib 包。例程中使用的版本是 lwip-1.4.1。

2.1.3.2 LwIP 的主要特性

1. 支持 ARP 协议（以太网地址解析协议）。
2. 支持 ICMP 协议（控制报文协议），用于网络的调试与维护。
3. 支持 IGMP 协议（互联网组管理协议），可以实现多播数据的接收。

4. 支持 UDP 协议（用户数据报协议）。

5. 支持 TCP 协议（传输控制协议），包括阻塞控制、RTT 估算、快速恢复和快速转发。

6. 支持 PPP 协议（点对点通信协议），支持 PPPoE。

7. 支持 DNS（域名解析）。

8. 支持 DHCP 协议，动态分配 IP 地址。

9. 支持 IP 协议，包括 IPv4、IPv6 协议，支持 IP 分片与重装功能，多网络接口下的数据包转发。

10. 支持 SNMP 协议（简单网络管理协议）。

11. 支持 AUTOIP，自动 IP 地址配置。

12. 提供专门的内部回调接口（Raw API），用于提高应用程序性能。

13. 提供可选择的 Socket API、NETCONN API（在多线程情况下使用）。

2.1.3.3 LwIP 的编程接口

LwIP 提供了三种编程接口，分别为 RAW/Callback API、NETCONN API、SOCKET API。SOCKET API 的易用性最高相对执行效率也最低，而 RAW/Callback API 的易用性最低但执行效率最高，需要用户根据实际情况进行选择。

2.2 APM32 的 ETH 外设

以太网外设包括带专用 DMA 控制器的 MAC 802.3。它支持默认情况下使用的 MII 和 RMII，并通过选择位切换。还包括用于与外部 PHY 通信的 SMI。通过一配置寄存器可以选择 MAC 控制器和 DMA 控制器的模式和功能。

在发送数据时，先将数据由系统存储器以 DMA 的方式发送至 TX FIFO 缓冲后通过 MAC 内核发送。通过线路接收的以太网帧经过 DMA 发送到系统存储器之前由 RX FIFO 存储。

2.2.1 站管理接口（SMI）

SMI 支持访问 32 个 PHY，应用程序通过 2 线时钟和数据线从 32 个 PHY 中选择一个 PHY，然后访问任意 PHY 寄存器。任意给定时间内只能对一个 PHY 中的一个寄存器进行寻址。

SMI 写操作：当 MAC_ADDR 的 MB 位和 MW 位被应用程序置 1 时，SMI 通过发送 PHY 地址、PHY 中的寄存器地址以及写入数据来触发 PHY 寄存器的写操作。当执行写操作时，应用程序不能修改 MAC_ADDR 和 MAC_DATA 寄存器。完成写操作后，SMI 将复位 MB 位。

SMI 读操作：当设置 MAC_ADDR 中的 MB 位、且 MW 位清零时，SMI 将通过发送 PHY 地址和 PHY 中的寄存器地址来触发 PHY 寄存器的读操作。当执行读操作时，应用程序不能修改

MAC_ADDR 和 MAC_DATA 寄存器。完成读操作后，SMI 将复位 MB 位，然后将从 PHY 中读取的数据更新到 MAC_DATA 寄存器。

2.2.2 介质独立接口（MII）

MII 定义了 MAC 子层与 PHY 在 10Mbit/s 和 100Mbit/s 的数据传输速率下的互连。信号如下：

MII_TX_EN：发送使能信号，MAC 当前正针对 MII 发送半字节

MII_RX_DV：数据接收有效信号，PHY 当前正针对 MII 接收已恢复并解码的半字节

MII_TXD[3:0]：数据发送信号

MII_RXD[3:0]：数据接收信号

MII_RX_ER：接收错误信号

MII_TX_CLK：连续时钟信号，为 TX 数据传输提供参考时序

MII_RX_CLK：连续时钟信号，为 RX 数据传输提供参考时序

MII_CRS：载波侦听信号

MII_COL：冲突检测信号

MII 时钟源：必须向外部 PHY 提供 25MHz 时钟才能生成 TX_CLK 和 RX_CLK 时钟信号，通过 MCO 引脚输出该信号。此时必须配置 PLL 倍频才可以通过 25MHz 外部石英晶体在 MCO 引脚上获得所需频率。

2.2.3 精简介质独立接口（RMII）

RMII 降低了以太网外设与外部 PHY 在 10/100Mbit/s 下微控制器的引脚数。根据 IEEE 802.3u 标准，MII 有 16 个数据和控制信号的引脚。RMII 将引脚数减少为 7 个。

RMII 在 MAC 和 PHY 之间实例化。有助于将 MAC 的 MII 转换为 RMII。RMII 具有以下特性：

单独的 2 位宽的发送和接收数据路径

10-Mbit/s 和 100-Mbit/s 的运行速率

参考时钟为 50MHz

从外部提供相同的参考时钟给 MAC 和外部以太网 PHY

RMII 时钟源：使用外部 50MHz 时钟或嵌入式 PLL 生成 50MHz 频率信号驱动 PHY。

2.3 硬件设计

开发板使用 APM32F407 控制器通过 RMII 接口和 SMI 接口与 LAN8720A 以太网 PHY 进行连接。

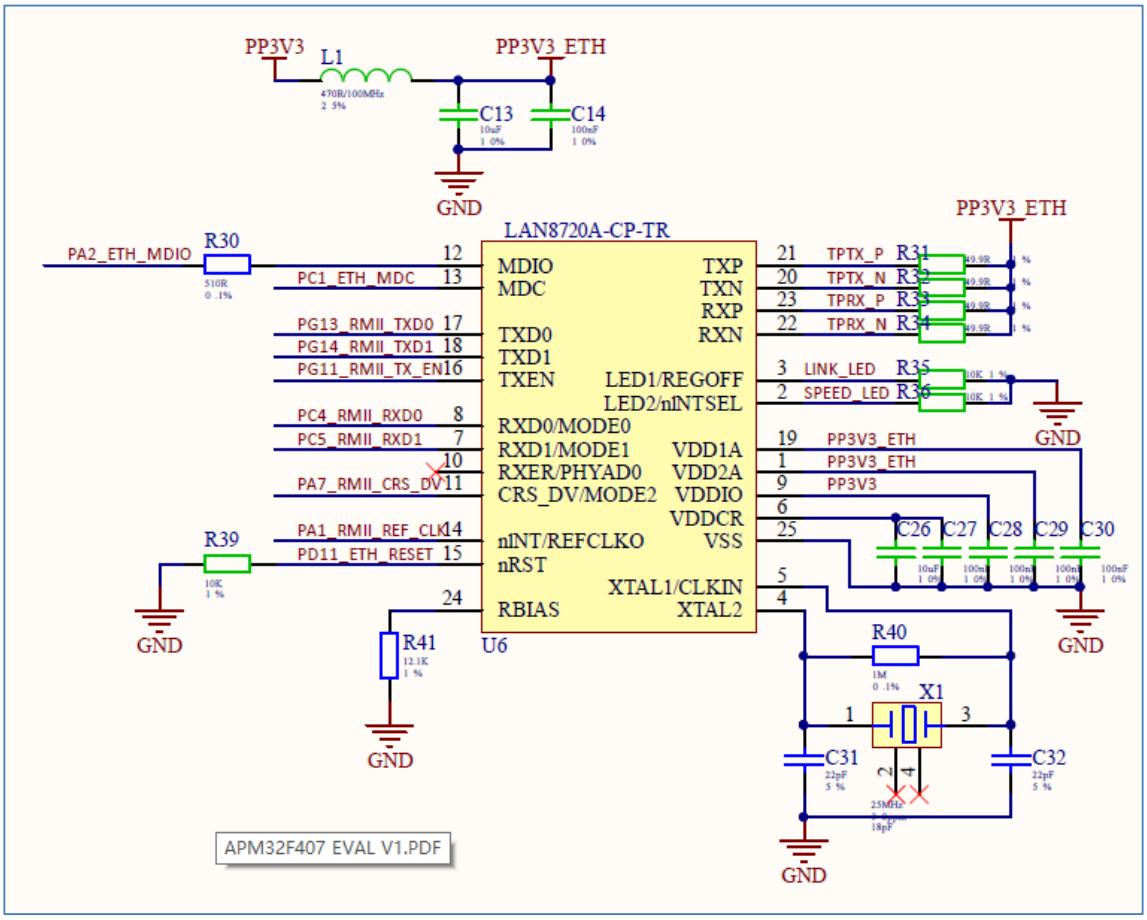


图 1 LAN8720A 硬件电路

通过接下拉电阻把 nINTSEL 引脚设置为低电平，从而使能 nINT/REFCLKO 引脚的输出功能为 RMII 接口中 REF_CLK 信号线提供时钟信号，硬件上 XTAL1 与 XTAL2 之间接入提供 25MHz 时钟，经过 LAN8720A 内部 PLL 电路陪频后使得 nINT/REFCLKO 引脚的输出的时钟信号为 50MHz 时钟。

3 移植 LwIP

本章主要介绍如何把 LwIP 移植到无操作系统的 APM32F407 裸机工程上并展示 ping 指令的实验演示。

3.1 PHY 底层驱动

因为标准库中并没有 ETH 外设的相关文件，因此首先需要在裸机工程中添加 ETH 驱动库文件 apm32f4xx_eth.c 和 apm32f4xx_eth.h 用于实现 ETH 驱动。

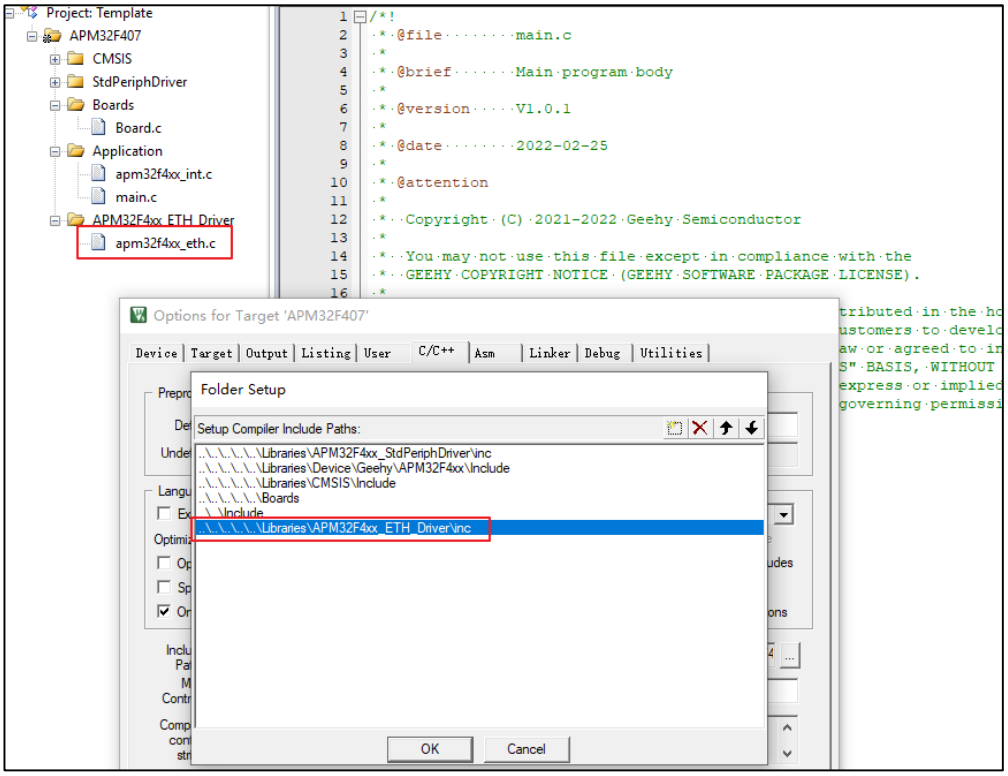


图 2 添加驱动库

新建底层驱动文件 board_LAN8720A.c 和 board_LAN8720A.h 实现对 PHY 相关驱动，初始化以太网所使用 RMII 接口的 GPIO 以及完成 ETH 的 MAC 和 DMA 的配置。最后通过在 main 函数中调用 ConfigEthernet( )完成对网卡的驱动。

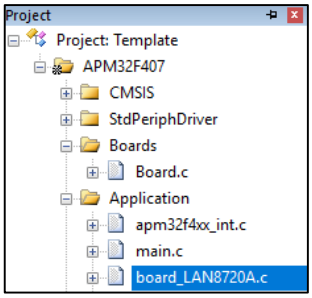


图 3 新建底层驱动文件

3.2 添加 LwIP 源文件

在已写好以太网 PHY 驱动的工程中新建分组 lwip 添加以下文件进人工程，无需修改。

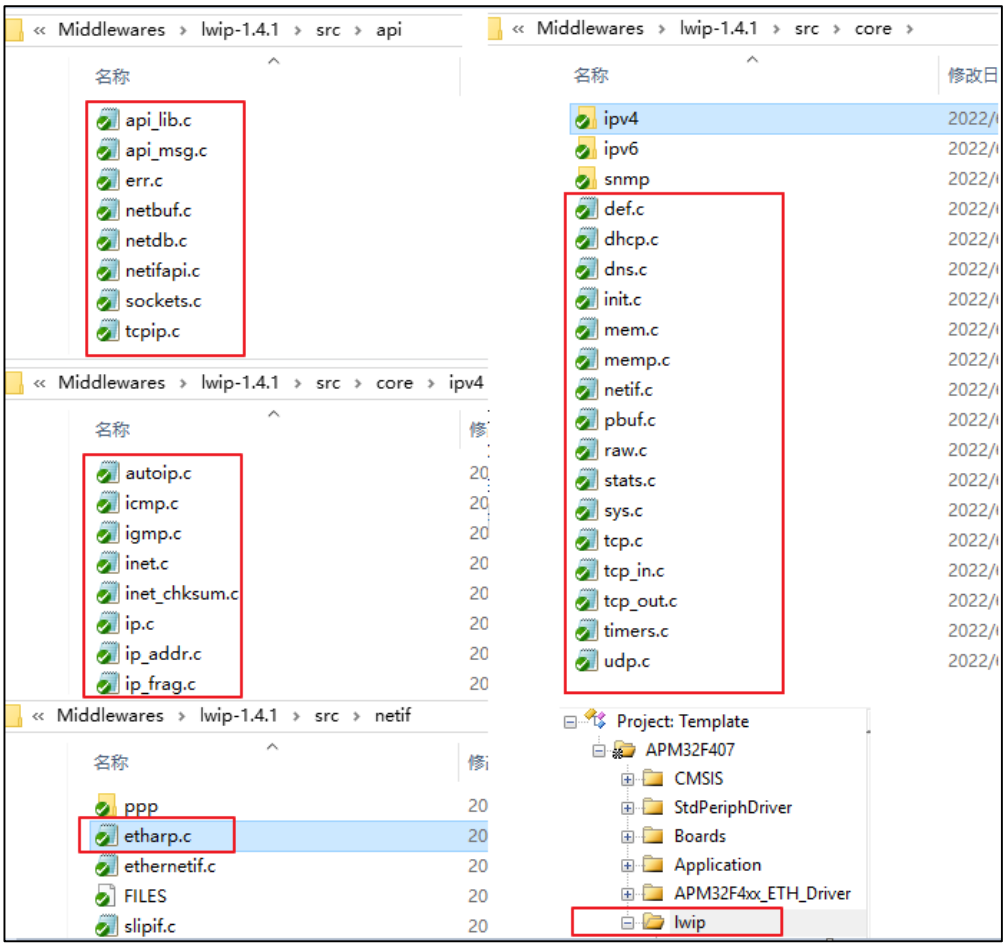


图 4 LWIP 源文件文件

3.3 移植头文件

移植完 LWIP 源文件后，我们的工程还需要一些相应的头文件支持把 LWIP 网站下载的 contrib-1.4.1\ports\old\rtxc\include\arch 中的整改 arch 文件夹复制到当前工程中并包含。

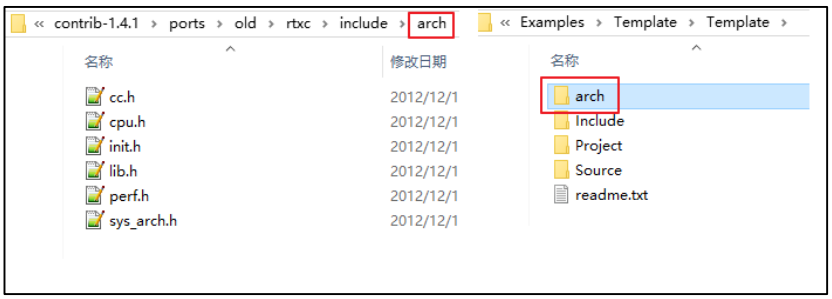
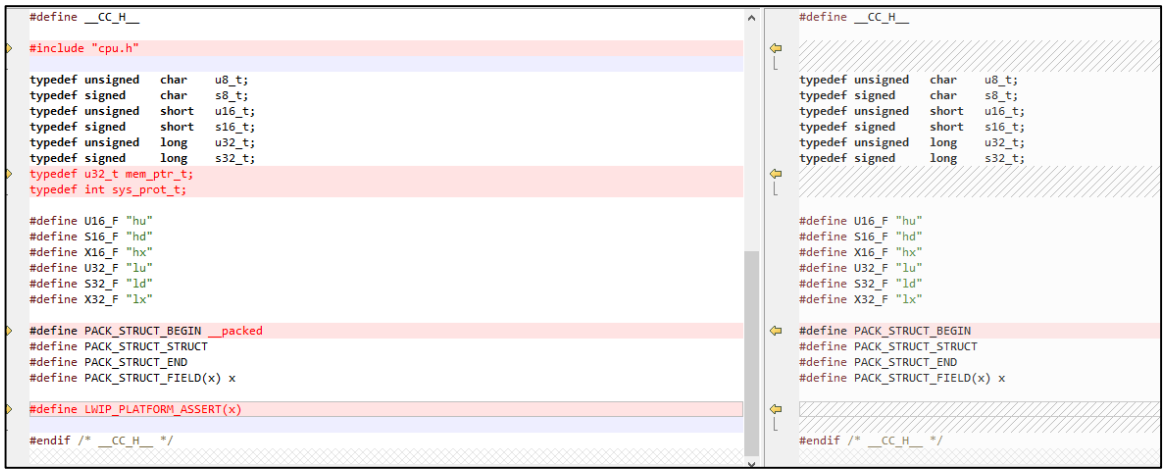


图 5 LWIP 头文件

因为我们使用 Keil 开发工具，需要对其中的 cc.h 文件进行简单修改，左边为修改后。



```
#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__ */
```

图 6 cc.h 文件

cc.h 文件中包含处理器相关的变量类型、数据结构及字节对齐的相关宏。U16_F、S16_F、X16_F 等等一系列名称用于 LwIP 的调试函数的调试信息输出格式。

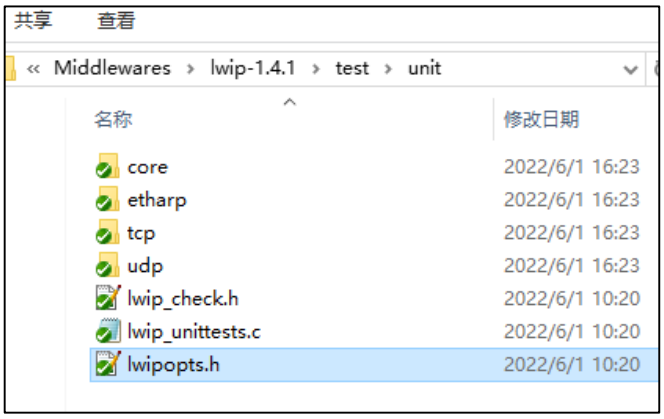


图 7 添加 lwipopts.h 文件

接着还需要对 LwIP 功能选项进行配置，把路径 lwip-1.4.1\test\unit\下的文件 lwipopts.h 拷贝到工程，进行修改，lwipopts.h 头文件可以对剪切 LwIP 功能进行裁剪，比如有无操作系统、内存空间分配、存储池分配、TCP 功能、UDP 功能选择，编程接口的使能等等。如果用户没有在 lwipopts.h 文件进行配置，那么 LwIP 就会使用 opt.h 默认的参数。

3.4 移植网卡驱动文件

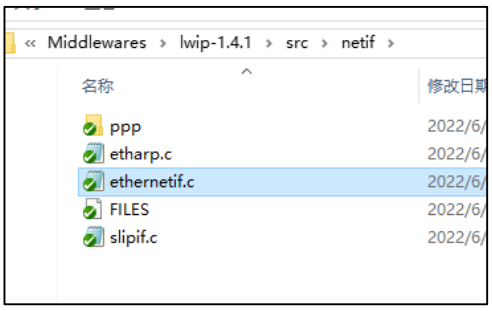


图 8 移植网卡驱动文件

lwip-1.4.1 中的 ethernetif.c 文件即为底层接口的驱动的模版，存放 LwIP 与 ETH 外设网络接口函数，在网卡接收或者发送数据的时候就会被调用，我们把此模块复制到工程中并根据使用到的网卡实现方式进行修改后 LWIP 的移植就基本完成了。

主要需要用户实现的函数有 low_level_init、low_level_output 和 low_level_input 分别用于初始化并使能 MAC 和 DMA、物理层发送数据和物理层接收数据。

3.5 初始化协议栈

需要添加和修改的文件已经准备好了，在使用到 LWIP 协议栈之前，还需要先初始化 LWIP。

```
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);
```

首先完成内存管理的初始化，通过 mem_init 和 memp_init 完成对内存堆和内存池的初始化。接着通过 IP4_ADDR 对变量 ipaddr、netmask 和 gw 进行一个初始化，设置为本地 IP 地址、子网掩码和网关的地址，再使用 netif_add 添加以太网设备，把 IP 地址、子网掩码、网关、网卡设备初始化函数、太网帧接收函数的地址传入 UserNetif 变量，完成网卡的注册。并通过功能函数 netif_set_default 把网卡 UserNetif 设置为默认的网络通信设备。最后调用 netif_set_up 启动网卡

就完成了 LWIP 的初始化了。需要注意的是 IP 地址的设置必须已路由器处于同一网关。比如我所使用的电脑 IP 为 192.168.73.122。就需要把网关设置为 192.168.73.1，掩码设置为 255.255.255.0，IP 地址为 192.168.73.xx。

3.6 配置 LwIP 时基

在整个 LWIP 内核初始化后想要协议栈正常的运作起来，还需要给 lwip 配置一个时基，使内核可以处理各种定时事件如 TCP 定时、ARP 定时任务，

采用 SysTick 作为 LwIP 的时基定时器，1ms 触发一次中断，对全局变量变量 ETHTimer 进行加 1，LwIP 通过两次获取的时间就能判断是否有超时，从而处理对应的事件。

同时 LwIP 中也定义了一个功能函数用于实现定时事件 sys_check_timeouts()。如果要使用到 sys_check_timeouts()函数需要，在 lwipopts.h 头文件中设置 NO_SYS_NO_TIMERS 为 0 实现支持 sys_timeout 功能，并实现功能函数 sys_now()，在函数中返回 ETHTimer 获取当前的 tick 值。

3.7 数据包的获取

至此，已经可以使用开发板获取网络的数据包了，一般使用轮询或者中断处理的方式，获取数据包。下面将举例在当前工程中通过轮询的方式进行数据包的获取，展示 ping 指令的实验现象，通过轮询的方式，只需要在 mian 函数中周期性使用 ETH_ReadRxPacketSize 判断是否有数据包到达，并在收到数据包时调用 ethernetif_input 接收函数即可。

3.8 实验现象

把工程编译并且将下载到开发板后，通过网线连接开发板和电脑并通过电脑的 CMD 控制台，在命令行中执行 ipconfig，确定电脑的 IP 地址网关为 192.168.73.1，并执行 ping 192.168.73.22，看到如下图实验现象即可确定我们的移植工作结束了网卡驱动与协议栈可以正常工作。

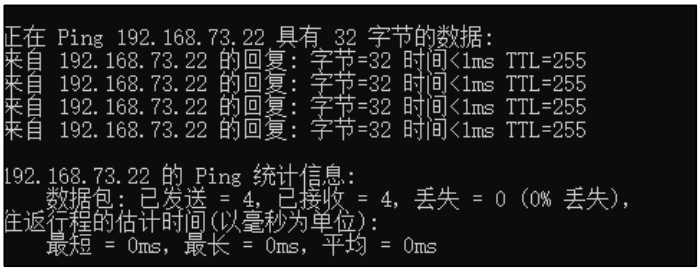


图 9 实验现象

4 版本历史

表 1 文件版本历史

日期	版本	变更历史
2022.06.23	1.0	新建

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“TM”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，除非在极海销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及／或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册相关信息而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失）。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。