HDSC 华大半导体 HUADA SEMICONDUCTOR

32 位微控制器

HC32F4A0 系列的以太网 LWIP 协议栈移植

本产品支持芯片系列如下

F 系列	**HC32F4A0**

目录

1 摘要

本篇应用笔记主要介绍 HC32F4A0 系列的以太网 LwIP 协议栈移植，并演示基于 lwip 的 http 服务器功能。

2 LwIP 协议栈

LwIP 为免费 TCP/IP 协议栈，由 AdamDunkels 在瑞典计算机科学院（SICS）开发，由修正的 BSD 许可授权。LwIP 实现的侧重点为在全面保持 TCP/IP 协议栈的同时，尽可能的减少 RAM 的使用。这使得 LwIP 特别适合在嵌入式系统中使用。

LwIP 具有下列协议：

- IPv4 和 IPv6（网际协议 v4 和 v6）
- ICMP（互联网控制消息协议），用于网络维护和调试
- IGMP（互联网组管理协议），用于多播流量的管理
- UDP（用户数据报协议）
- TCP（传输控制协议）
- DNS（域名服务器）
- SNMP（简单网络管理协议）
- DHCP（动态主机配置协议）
- PPP（点到点协议）
- ARP（地址解析协议）

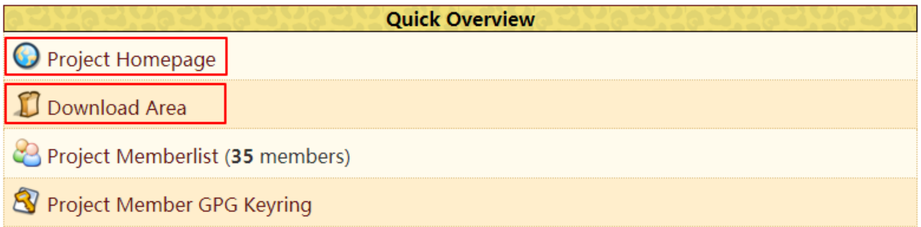
2.1 LwIP 目录结构

2.1.1 获取 LwIP 源代码

通过下面的链接打开 LwIP 的项目主页：

http://savannah.nongnu.org/projects/lwip/

主页对 LwIP 进行了简单的介绍，更多关于 LwIP 的信息需要打开主页上对应的链接去查看。其中“Project Homepage”链接会打开一个 LwIP 的详细说明文档，而“Download Area”链接会打开一个 LwIP 所有版本的源代码仓库。



Quick Overview

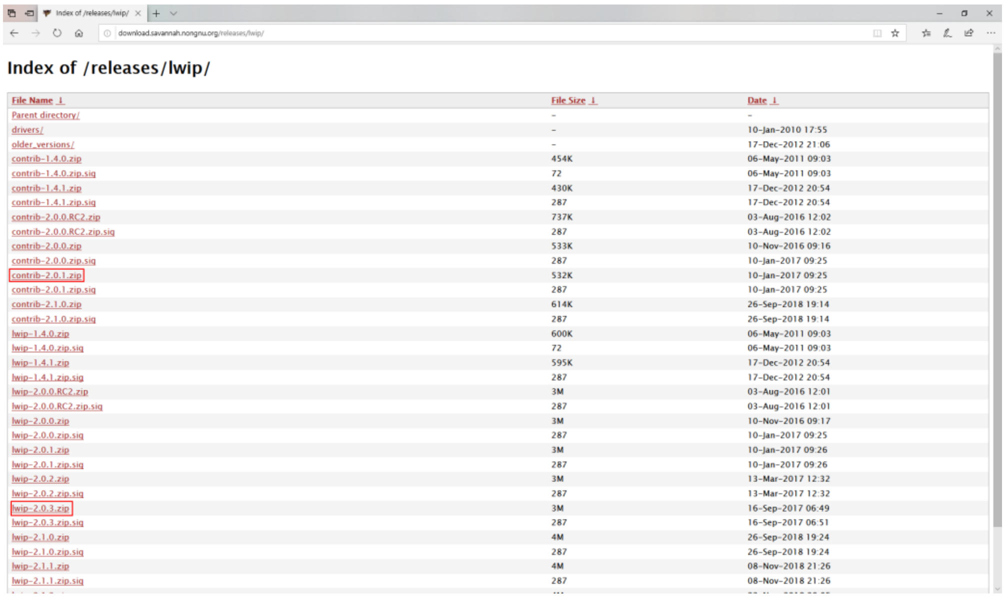
Project Homepage

Download Area

Project Memberlist (**35** members)

Project Member GPG Keyring

点击“Download Area”进入源代码仓库网页，样例演示使用的是 lwip-2.0.3 版本的源代码，点击“lwip-2.0.3.zip”下载该版本源代码；而移植需要用到 contrib 里面一些文件，所以需要对应点击“contrib-2.0.1.zip”下载该版本的源代码，如下图：



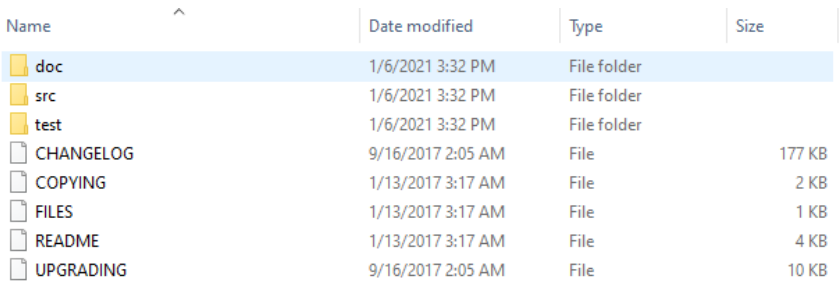
Index of /releases/lwip/

File Name ↓	File Size ↓	Date ↓
Parent directory/	-	-
drivers/	-	10-Jan-2010 17:55
older_versions/	-	17-Dec-2012 21:06
contrib-1.4.0.zip	454K	06-May-2011 09:03
contrib-1.4.0.zip.sig	72	06-May-2011 09:03
contrib-1.4.1.zip	430K	17-Dec-2012 20:54
contrib-1.4.1.zip.sig	287	17-Dec-2012 20:54
contrib-2.0.0.RC2.zip	737K	03-Aug-2016 12:02
contrib-2.0.0.RC2.zip.sig	287	03-Aug-2016 12:02
contrib-2.0.0.zip	533K	10-Nov-2016 09:16
contrib-2.0.0.zip.sig	287	10-Jan-2017 09:25
contrib-2.0.1.zip	532K	10-Jan-2017 09:25
contrib-2.0.1.zip.sig	287	10-Jan-2017 09:25
contrib-2.1.0.zip	614K	26-Sep-2018 19:14
contrib-2.1.0.zip.sig	287	26-Sep-2018 19:14
lwip-1.4.0.zip	600K	06-May-2011 09:03
lwip-1.4.0.zip.sig	72	06-May-2011 09:03
lwip-1.4.1.zip	595K	17-Dec-2012 20:54
lwip-1.4.1.zip.sig	287	17-Dec-2012 20:54
lwip-2.0.0.RC2.zip	3M	03-Aug-2016 12:01
lwip-2.0.0.RC2.zip.sig	287	03-Aug-2016 12:01
lwip-2.0.0.zip	3M	10-Nov-2016 09:17
lwip-2.0.0.zip.sig	287	10-Jan-2017 09:25
lwip-2.0.1.zip	3M	10-Jan-2017 09:26
lwip-2.0.1.zip.sig	287	10-Jan-2017 09:26
lwip-2.0.2.zip	3M	13-Mar-2017 12:32
lwip-2.0.2.zip.sig	287	13-Mar-2017 12:32
lwip-2.0.3.zip	3M	16-Sep-2017 06:49
lwip-2.0.3.zip.sig	287	16-Sep-2017 06:51
lwip-2.1.0.zip	4M	26-Sep-2018 19:24
lwip-2.1.0.zip.sig	287	26-Sep-2018 19:24
lwip-2.1.1.zip	4M	08-Nov-2018 21:26
lwip-2.1.1.zip.sig	287	08-Nov-2018 21:26

其中 lwip-2.0.3.zip 压缩包中主要是 LwIP 内核的源码文件，而 contrib-2.0.1.zip 压缩包中主要是移植和应用 LwIP 的一些 example，不属于 LwIP 内核的一部分。

2.1.2 LwIP 文件说明

解压下载的两个压缩包：lwip-2.0.3.zip 和 contrib-2.0.1.zip，得到对应的文件夹，打开“lwip-2.0.3”文件夹，如下图所示：



Name	Date modified	Type	Size
doc	1/6/2021 3:32 PM	File folder	
src	1/6/2021 3:32 PM	File folder	
test	1/6/2021 3:32 PM	File folder	
CHANGELOG	9/16/2017 2:05 AM	File	177 KB
COPYING	1/13/2017 3:17 AM	File	2 KB
FILES	1/13/2017 3:17 AM	File	1 KB
README	1/13/2017 3:17 AM	File	4 KB
UPGRADING	9/16/2017 2:05 AM	File	10 KB

lwip-2.0.3 文件夹下的文件功能简介如下：

1) doc 文件夹： LwIP 的应用和移植指南

2) src 文件夹： lwIP 协议栈的源代码

3) test 文件夹： 一些用来测试 LwIP 内核性能的源码

4) CHANGELOG： 记录 LwIP 在版本升级过程中源代码发生的变化

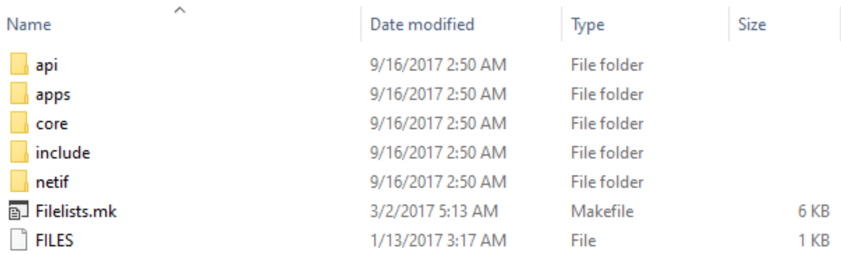
5) COPYING： LwIP 的 license

6) FILES： 介绍当前目录下的目录信息

7) README： LwIP 的简单介绍

8) UPGRADING： 记录 LwIP 每个版本的更新，对用户使用和移植 LwIP 造成的影响

打开 src 文件夹（LwIP 源码文件所在的目录），如下图所示：



Name	Date modified	Type	Size
api	9/16/2017 2:50 AM	File folder	
apps	9/16/2017 2:50 AM	File folder	
core	9/16/2017 2:50 AM	File folder	
include	9/16/2017 2:50 AM	File folder	
netif	9/16/2017 2:50 AM	File folder	
Filelists.mk	3/2/2017 5:13 AM	Makefile	6 KB
FILES	1/13/2017 3:17 AM	File	1 KB

源码文件夹说明如下：

1) api 文件夹： 包含高级（Netconn 和 Socket）API 的源文件，只有在操作系统的环境中才能被编译；使用低级（Call-back/Raw）API，则不需要。

2) apps 文件夹： 包含使用低级（Call-back/Raw）API 编程的高层应用程序，包括常见的应用程序，如 httpd、mqtt、tftp 等。

3) core 文件夹： 包含 TCP/IP 协议栈的核心源代码：协议实现、内存和缓冲区管理、低级（Call-back/Raw）API。

4) include 文件夹： 包含 LwIP 所有模块对应的头文件。

5) netif 文件夹： 包含通用网络接口设备驱动程序，即网卡移植相关的文件模板。

2.2 LwIP 的三种应用编程接口

LwIP 提供了三种应用编程接口，分别为 Call-back/RawAPI、NetconnAPI、SocketAPI。它们的易用性从左到右依次提高，而执行效率从左到右依次降低，用户可以根据实际情况，权衡利弊，选择合适的 API 进行网络应用程序的开发。

2.2.1 Call-back/RawAPI

Call-back/RawAPI 为原始的 LwIPAPI，它通过事件回调机制进行应用开发，该 API 提供了最好的性能和优化的代码长度，但增加了应用开发的复杂性。

2.2.2 NetconnAPI

NetconnAPI 为高层有序 API，需要实时操作系统（RTOS）的支持（提供进程间通讯的方法），其执行模型基于典型的阻塞式打开-读-写-关闭机制；若要正常工作，此 API 必须处于多线程工作模式，该模式需为 LwIPTCP/IP 栈实现专用线程，并/或为应用实现多个线程。

2.2.3 SocketAPI

SocketAPI 是类似 Berkeley 的套接字 API，它是有序 API，在内部构建于 NetconnAPI 之上。

3 ETH 简介

以太网 MAC 控制器（ETHMAC）用于在以太网网络中按照 IEEE802.3-2002 标准发送和接收数据，有多种应用领域，如交换机、网络接口卡等。该 MAC 控制器支持与外部物理层（PHY）相连的两个工业标准接口：介质独立接口（MII）（在 IEEE802.3 规范中定义）和简化介质独立接口（RMII）。

主要遵循以下协议规范：

- IEEE802.3-2002，用于以太网 MAC
- IEEE1588-2008 标准，用于规定联网时钟同步
- AMBA2.0，用于 AHB 主/从端口
- RMII 接口规范

3.1 基本框图

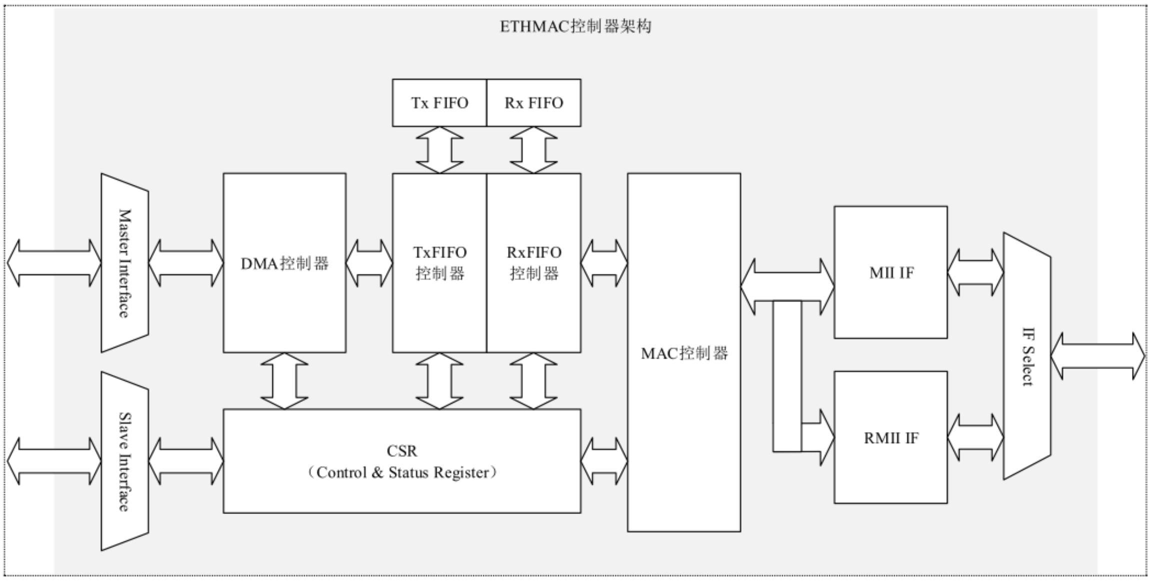


图 1 以太网 MAC 控制器架构图

从图中可以看出，以太网 MAC 控制器的内核部分包括 AHBMaster 接口、AHBSlave 接口、RxFIFO、TxFIFO 和 DMA 控制器、MAC 控制器、接口控制逻辑等。DMA 控制器通过 AHB 主从接口将 MAC 内核和系统存储器相连。AHBMaster 接口用于数据传输，AHBSlave 接口用于该控制器的基本寄存器访问配置等。

在发送数据时，首先将数据从系统存储器由 DMA 控制器送至 TxFIFO 进行缓冲，经 MAC 控制器的一系列处理（COE 控制、PTP 控制、MMC 控制等）后通过 MII 接口或 RMII 接口送至外部 PHY。

在接收数据时，经过 MII 接口或 RMII 接口接收进来的数据，经 MAC 控制器的一系列处理（COE 控制、PTP 控制、MMC 控制等）后送至 RxFIFO，DMA 控制器再将 RxFIFO 的数据传送到系统存储器。

4 PHY 简介

RTL8201F-VB-CG、RTL8201FL-VB-CG 和 RTL8201FN-VB-CG 是单芯片/单端口 10/100Mbps 以太网 PHY，支持：

- MII（媒体独立接口）
- RMII（精简介质独立接口）

RTL8201F/FL/FN 实现所有 10/100M 以太网物理层功能，包括物理编码子层(PCS)、物理介质附件(PMA)、双绞线物理介质相关子层(TP-PMD)、10BASE-TX 编码器/解码器和双绞线介质访问单元(TPMAU)。RTL8201F/FL/FN 支持自动 MDIX。

支持 PECL(伪发射器耦合逻辑)接口与外部 100BASE-FX 光纤收发器连接。该芯片采用先进的 CMOS 工艺，以满足低电压和低功耗的要求。该芯片采用片上 DSP(数字信号处理)技术，在所有工作条件下都能提供优异的性能。

4.1 主要特性

- 支持 IEEE802.3az-2010(EEE)
- 符合 10BASE-TIEEE802.3 标准
- 支持 MII 模式和 RMII 模式
- 双绞线或光纤模式输出
- 支持自动协商
- 支持中断功能
- 支持局域网唤醒(WOL)
- 支持 25MHz 外部晶体或 OSC
- 支持 50MHz 外部 OSC 时钟输入
- 为 MAC 提供 50MHz 时钟源

4.2 基本框图

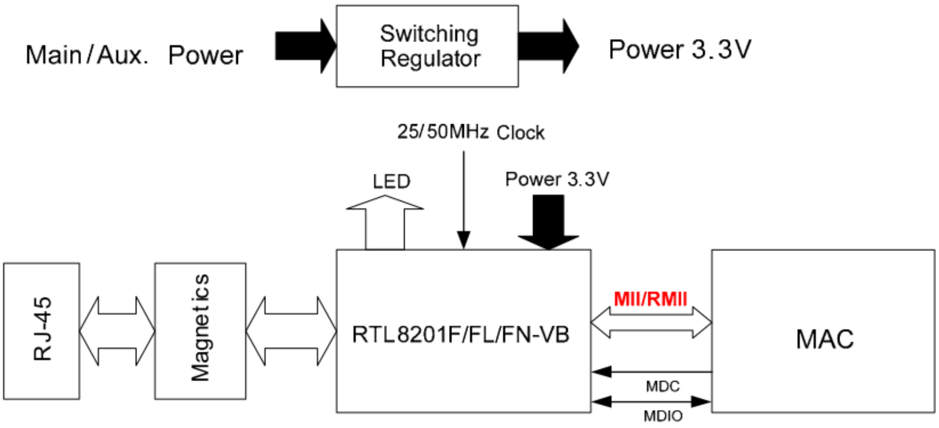


图 2 应用程序关系图

5 LwIP 移植

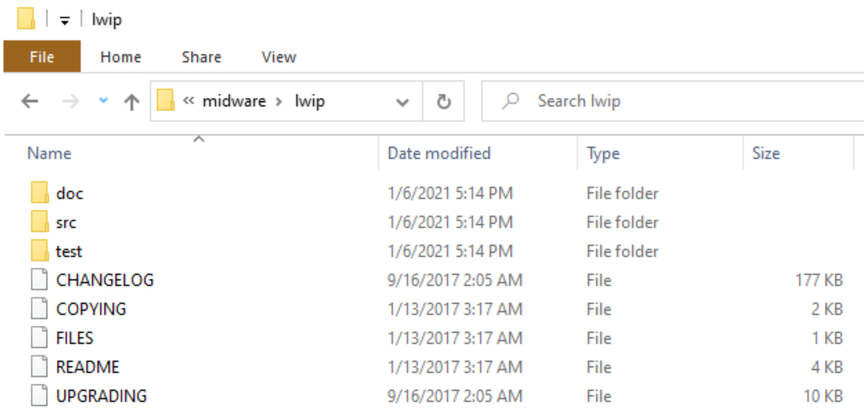
LwIP 不仅能在裸机上运行，也能在操作系统环境下运行，而且在操作系统环境下，用户能使用 NetconnAPI 与 SocketAPI 编程，相比 Call-back/RawAPI 编程会更加简便。

本篇笔记主要介绍的是裸机环境下移植并运行 LwIP 协议栈。

5.1 添加 LwIP 到工程

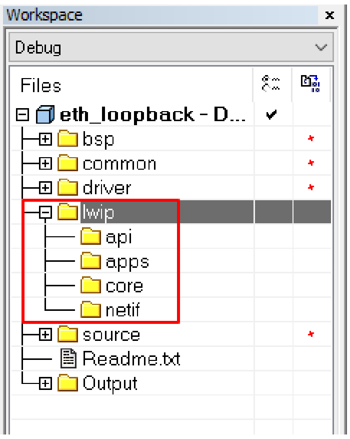
打开 DDL 中“..\example\eth\eth_loopback”样例，在此工程基础上添加 LwIP 组件，并修改 main 函数的用户程序即可。

1) 将 LwIP 源码文件夹“lwip-2.0.3”整个复制到 DDL 根目录结构中的“midware”文件夹下面，并修改文件夹名为“lwip”，如下图：

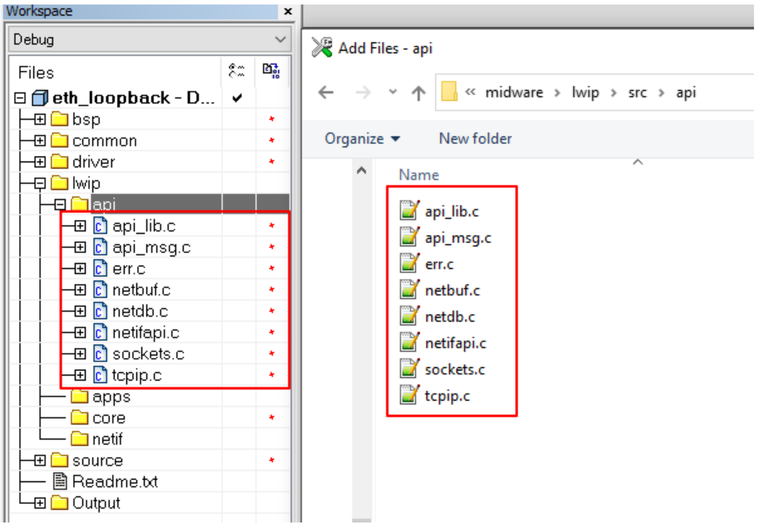


Name	Date modified	Type	Size
doc	1/6/2021 5:14 PM	File folder	
src	1/6/2021 5:14 PM	File folder	
test	1/6/2021 5:14 PM	File folder	
CHANGELOG	9/16/2017 2:05 AM	File	177 KB
COPYING	1/13/2017 3:17 AM	File	2 KB
FILES	1/13/2017 3:17 AM	File	1 KB
README	1/13/2017 3:17 AM	File	4 KB
UPGRADING	9/16/2017 2:05 AM	File	10 KB

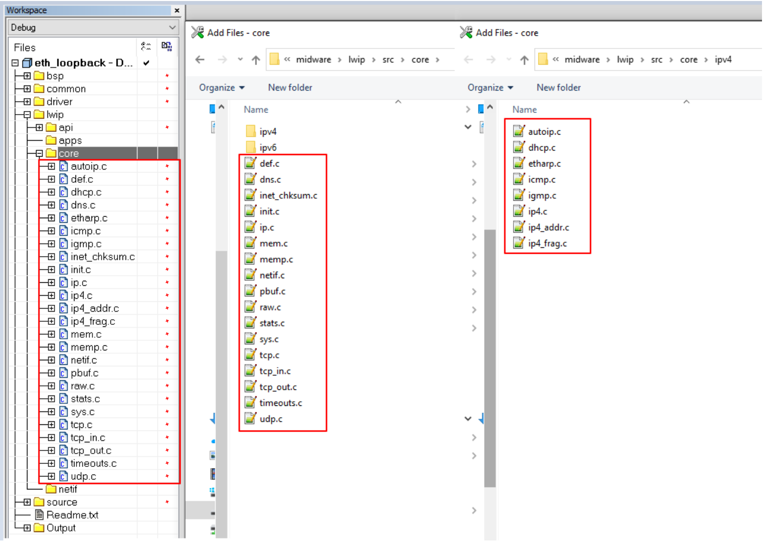
2) 使用 IAR 软件打开工程文件(..\eth_loopback\EWARM\eth_loopback.eww)，并添加 lwip 组和 4 个子组，组名和源码中的文件夹名相对应，如下图：



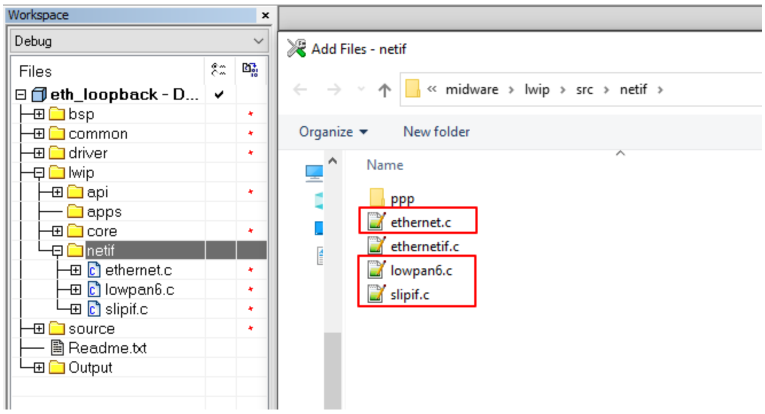
3) 向 api 分组添加“..\lwip\src\api”文件夹下面所有的 C 文件，添加完成后如下图：



4) 向 core 分组添加“..\lwip\src\core”文件夹和“..\lwip\src\core\ipv4”文件夹下面所有的 C 文件，添加完成后如下图：

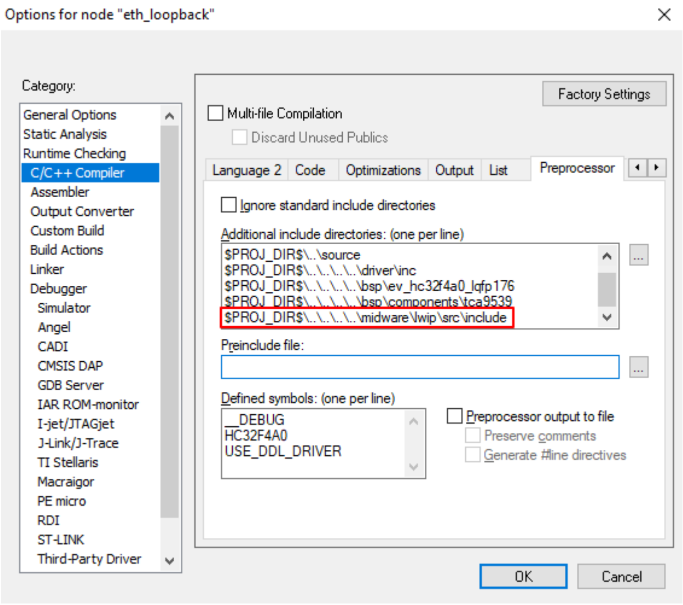


5) 向 netif 分组添加“..\lwip\src\netif”文件夹下面（除 ethernetif.c 外）所有的 C 文件，添加完成后如下图：



因为工程本身已经包含了 ethernetif.c，但需要根据“..\lwip\src\netif\ethernetif.c”文件中的接口标准对已有的 ethernetif.c 文件内容进行修改。

6) 添加 LwIP 头文件路径到工程的 include 选项中，如下图：



□ Multi-file Compilation

□ Discard Unused Publics

□ Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$\..\source
$PROJ_DIR$\..\..\..\..\driver\inc
$PROJ_DIR$\..\..\..\..\bsp\ev_hc32f4a0_lqfp176
$PROJ_DIR$\..\..\..\..\bsp\components\tca9539
$PROJ_DIR$\..\..\..\..\midware\lwip\src\include
```

Preinclude file:

Defined symbols: (one per line)

```
__DEBUG
HC32F4A0
USE_DDL_DRIVER
```

□ Preprocessor output to file

□ Preserve comments

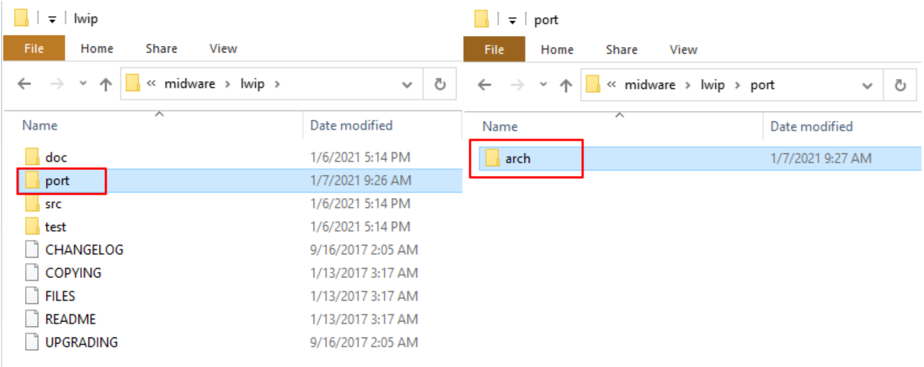
□ Generate #line directives

7) 至此，LwIP 的源代码已经基本添加到工程，但还无法正常编译和运行。

5.2 添加必要的头文件

在前面工程的基础上，要解决编译的问题，还需要一些头文件的支持，分别是 lwipopts.h、cc.h、perf.h、bpstruct.h、epstruct.h 等，其中 lwipopts.h 用于配置 LwIP 的相关参数。

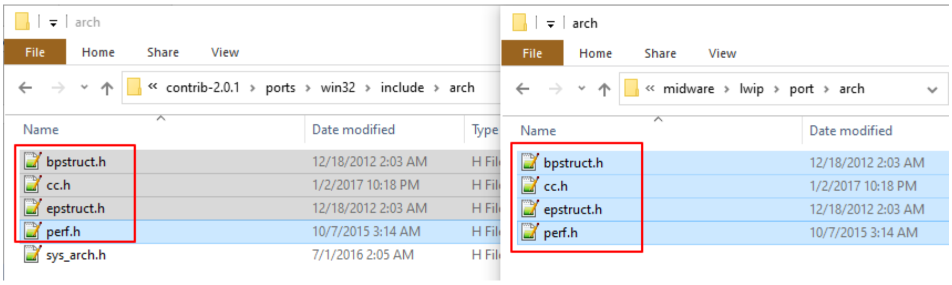
1) 首先在 DDL 目录结构中的“..\midware\lwip”文件夹下创建文件夹“port”，然后在“port”文件夹下创建文件夹“arch”，如下图：



Name	Date modified
doc	1/6/2021 5:14 PM
port	1/7/2021 9:26 AM
src	1/6/2021 5:14 PM
test	1/6/2021 5:14 PM
CHANGELOG	9/16/2017 2:05 AM
COPYING	1/13/2017 3:17 AM
FILES	1/13/2017 3:17 AM
README	1/13/2017 3:17 AM
UPGRADING	9/16/2017 2:05 AM

Name	Date modified
arch	1/7/2021 9:27 AM

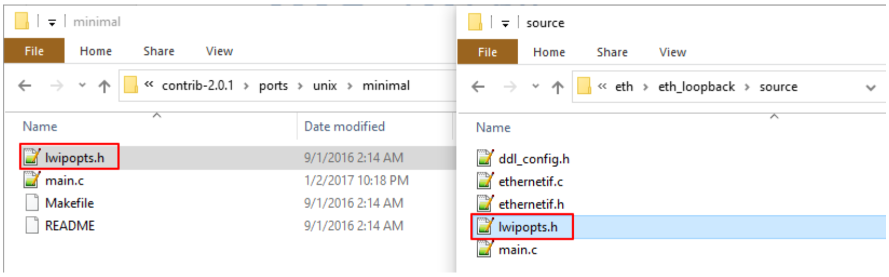
2) 打开之前解压的文件夹“contrib-2.0.1”，并将“contrib-2.0.1\ports\win32\include\arch”目录下的 cc.h、perf.h、bpstruct.h、epstruct.h 文件复制到 DLL 目录结构中的“..\midware\lwip\port\arch”文件夹下，如下图：


arch
File Home Share View
« contrib-2.0.1 › ports › win32 › include › arch
arch
File Home Share View
« midware › lwip › port › arch

Name	Date modified	Type
bpstruct.h	12/18/2012 2:03 AM	H Fil
cc.h	1/2/2017 10:18 PM	H Fil
epstruct.h	12/18/2012 2:03 AM	H Fil
perf.h	10/7/2015 3:14 AM	H Fil
sys_arch.h	7/1/2016 2:05 AM	H Fil

Name	Date modified
bpstruct.h	12/18/2012 2:03 AM
cc.h	1/2/2017 10:18 PM
epstruct.h	12/18/2012 2:03 AM
perf.h	10/7/2015 3:14 AM

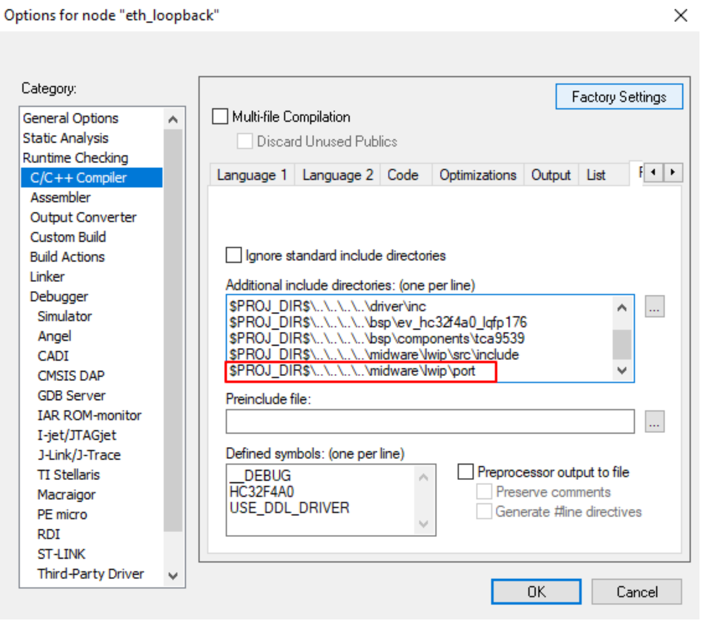
3) 再将“contrib-2.0.1\ports\unix\minimal”目录下的 lwipopts.h 文件复制到当前工程“..\eth\eth_loopback\source”的文件夹下，如下图：


minimal
File Home Share View
« contrib-2.0.1 › ports › unix › minimal
source
File Home Share View
« eth › eth_loopback › source

Name	Date modified
lwipopts.h	9/1/2016 2:14 AM
main.c	1/2/2017 10:18 PM
Makefile	9/1/2016 2:14 AM
README	9/1/2016 2:14 AM

Name
ddl_config.h
ethernetif.c
ethernetif.h
lwipopts.h
main.c

4) 将“port”文件夹路径到工程的 include 选项中，如下图：



□ Multi-file Compilation

□ Discard Unused Publics

□ Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\..\..\driver\inc
$PROJ_DIR$\..\..\..\..\bsp\ev_hc32f4a0_lqfp176
$PROJ_DIR$\..\..\..\..\bsp\components\tca9539
$PROJ_DIR$\..\..\..\..\midware\lwip\src\include
$PROJ_DIR$\..\..\..\..\midware\lwip\port
```

Preinclude file:

Defined symbols: (one per line)

```
__DEBUG
HC32F4A0
USE_DDL_DRIVER
```

□ Preprocessor output to file

□ Preserve comments

□ Generate #line directives

5) 因编译和运行 LwIP 的环境不同，所以需要修改相关的头文件；将“bpstruct.h”文件修改为如下：

```
#if defined(__IAR_SYSTEMS_ICC__)
#pragma pack(1)
#endif
```

6) 将“epstruct.h”文件修改为如下：

```
#if defined(__IAR_SYSTEMS_ICC__)
#pragma pack()
#endif
```

7) 将“cc.h”文件修改为如下：

```
#ifndef __CC_H__
#define __CC_H__

#include <stdlib.h>
#include <stdio.h>

typedef int sys_prot_t;

#define LWIP_PROVIDE_ERRNO

#if defined (__GNUC__) & !defined (__CC_ARM)

#define LWIP_TIMEVAL_PRIVATE          0
```

```
#include <sys/time.h>

#endif

#ifndef BYTE_ORDER
#define BYTE_ORDER                    LITTLE_ENDIAN
#endif

/* define compiler specific symbols */
#if defined (__ICCARM__)

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

#elif defined (__CC_ARM)

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

#elif defined (__GNUC__)

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

#endif

#define LWIP_PLATFORM_ASSERT(x)                                       \
do {                                                        \
   printf("Assertion \"%s\" failed at line %d in %s\n", x, __LINE__, __FILE__);   \
   } while(0)

/* Define random number generator function */
#define LWIP_RAND()                   ((u32_t)rand())

#endif /* __CC_H__ */
```

8) 至此，LwIP 源代码已经移植完成，可以顺利编译，但要正常运行还需要移植网卡驱动。

5.3 移植网卡驱动

根据“..\lwip\src\netif\ethernetif.c”文件中的函数接口可知网卡驱动最主要的就是根据不同的硬件重新实现这个 5 个函数，以太网函数接口说明如下表：

函数	说明
low_level_init	调用以太网驱动函数，初始化以太网外设
low_level_output	调用以太网驱动函数以发送以太网包
low_level_input	调用以太网驱动函数以接收以太网包
ethernetif_input	调用 low_level_input 接收数据包，然后将其提供给 LwIP 协议栈
ethernetif_init	初始化网络接口结构并调用 low_level_init 以初始化以太网外设

要让 LwIP 正常运行还需要一个时基，因其是由时基驱动的，以处理内核中各种定时事件，如 ARP 定时、TCP 定时等，因此需要根据硬件实际实现一个 sys_now()函数来获取系统的时钟，以毫秒为单位。

5.3.1 修改 ethernetif.c 文件

根据上述要求对工程现有的“ethernetif.c”文件进行修改，修改后如下：

```
#include "hc32_ddl.h"
#include "lwip/timeouts.h"
#include "netif/etharp.h"
#include "ethernetif.h"

/* Define those to better describe your network interface. */
#define IFNAME0                         'h'
#define IFNAME1                         'd'

/* Global Ethernet handle*/
static stc_eth_handle_t EthHandle;

/* Ethernet Tx DMA Descriptor */
__ALIGN_BEGIN static stc_eth_dma_desc_t EthDmaTxDscrTab[ETH_TXBUF_NUMBER];
/* Ethernet Rx DMA Descriptor */
__ALIGN_BEGIN static stc_eth_dma_desc_t EthDmaRxDscrTab[ETH_RXBUF_NUMBER];
/* Ethernet Transmit Buffer */
__ALIGN_BEGIN static uint8_t EthTxBuff[ETH_TXBUF_NUMBER][ETH_TXBUF_SIZE];
/* Ethernet Receive Buffer */
__ALIGN_BEGIN static uint8_t EthRxBuff[ETH_RXBUF_NUMBER][ETH_RXBUF_SIZE];

/**
```

```
 * @brief  Initializes the Ethernet GPIO.
 * @param  None
 * @retval None
 */
static void Ethernet_GpioInit(void)
{
    /* ETH_RST */
    BSP_IO_ConfigPortPin(EIO_PORT1, EIO_ETH_RST, EIO_DIR_OUT);
    BSP_IO_WritePortPin(EIO_PORT1, EIO_ETH_RST, (uint8_t)Disable);
    SysTick_Delay(PHY_HW_RESET_DELAY);
    BSP_IO_WritePortPin(EIO_PORT1, EIO_ETH_RST, (uint8_t)Enable);
    SysTick_Delay(PHY_HW_RESET_DELAY);

   /* Configure MII/RMII selection IO for ETH */
#ifdef ETH_INTERFACE_RMII
    /* Ethernet RMII pins configuration */
    /*
        ETH_SMI_MDIO ----------------> PA2
        ETH_SMI_MDC -----------------> PC1
        ETH_RMII_TX_EN --------------> PG11
        ETH_RMII_TXD0 ---------------> PG13
        ETH_RMII_TXD1 ---------------> PG14
        ETH_RMII_REF_CLK ------------> PA1
        ETH_RMII_CRS_DV -------------> PA7
        ETH_RMII_RXD0 ---------------> PC4
        ETH_RMII_RXD1 ---------------> PC5
        ETH_RMII_RX_ER --------------> PI10
    */
    /* Configure PA1, PA2 and PA7 */
    GPIO_SetFunc(GPIO_PORT_A, (GPIO_PIN_01 | GPIO_PIN_02 | GPIO_PIN_07),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PC1, PC4 and PC5 */
    GPIO_SetFunc(GPIO_PORT_C, (GPIO_PIN_01 | GPIO_PIN_04 | GPIO_PIN_05),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PG11, PG13 and PG14 */
    GPIO_SetFunc(GPIO_PORT_G, (GPIO_PIN_11 | GPIO_PIN_13 | GPIO_PIN_14),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PI10 */
    GPIO_SetFunc(GPIO_PORT_I, GPIO_PIN_10,
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
#else
    /* Ethernet MII pins configuration */
    /*
        ETH_SMI_MDIO ----------------> PA2
        ETH_SMI_MDC -----------------> PC1
        ETH_MII_TX_CLK --------------> PB6
        ETH_MII_TX_EN ---------------> PG11
        ETH_MII_TXD0 ----------------> PG13
        ETH_MII_TXD1 ----------------> PG14
        ETH_MII_TXD2 ----------------> PB9
        ETH_MII_TXD3 ----------------> PB8
        ETH_MII_RX_CLK --------------> PA1
        ETH_MII_RX_DV ---------------> PA7
        ETH_MII_RXD0 ----------------> PC4
        ETH_MII_RXD1 ----------------> PC5
        ETH_MII_RXD2 ----------------> PB0
        ETH_MII_RXD3 ----------------> PB1
        ETH_MII_RX_ER ---------------> PI10
        ETH_MII_CRS -----------------> PH2
        ETH_MII_COL -----------------> PH3
    */
    /* Configure PA1, PA2 and PA7 */
    GPIO_SetFunc(GPIO_PORT_A, (GPIO_PIN_01 | GPIO_PIN_02 | GPIO_PIN_07),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PB0, PB1, PB6, PB8 and PB9 */
    GPIO_SetFunc(GPIO_PORT_B, (GPIO_PIN_00 | GPIO_PIN_01 | GPIO_PIN_06 |
GPIO_PIN_08 | GPIO_PIN_09), GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PC1, PC4 and PC5 */
    GPIO_SetFunc(GPIO_PORT_C, (GPIO_PIN_01 | GPIO_PIN_04 | GPIO_PIN_05),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PG11, PG13 and PG14 */
    GPIO_SetFunc(GPIO_PORT_G, (GPIO_PIN_11 | GPIO_PIN_13 | GPIO_PIN_14),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PH2, PH3 */
    GPIO_SetFunc(GPIO_PORT_H, (GPIO_PIN_02 | GPIO_PIN_03),
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
    /* Configure PI10 */
    GPIO_SetFunc(GPIO_PORT_I, GPIO_PIN_10,
GPIO_FUNC_11_ETH ,PIN_SUBFUNC_DISABLE);
#endif
}

/**
 * @brief In this function, the hardware should be initialized.
 * Called from ethernetif_init().
 *
 * @param netif the already initialized lwip network interface structure
 *        for this ethernetif
 */
static void low_level_init(struct netif *netif)
{
    stc_eth_init_t stcEthInit;
    uint16_t u16RegVal;

    /* Init Ethernet GPIO */
    Ethernet_GpioInit();
    /* Enable ETH clock */
    PWC_Fcg1PeriphClockCmd(PWC_FCG1_ETHER, Enable);

    /* Reset ETHERNET */
    (void)ETH_DeInit();
    /* Configure structure initialization */
    (void)ETH_CommStructInit(&EthHandle.stcCommInit);
    (void)ETH_StructInit(&stcEthInit);
#ifdef ETH_INTERFACE_RMII
    EthHandle.stcCommInit.u32MediaInterface  = ETH_MAC_MEDIA_INTERFACE_RMII;
#else
    EthHandle.stcCommInit.u32MediaInterface  = ETH_MAC_MEDIA_INTERFACE_MII;
#endif
```

```
    /* Configure ethernet peripheral */
    if (Ok == ETH_Init(&EthHandle, &stcEthInit))
    {
        /* Set netif link flag */
        netif->flags |= NETIF_FLAG_LINK_UP;
    }

    /* Initialize Tx Descriptors list: Chain Mode */
    (void)ETH_DMA_TxDescListInit(&EthHandle, EthDmaTxDscrTab, &EthTxBuff[0][0],
ETH_TXBUF_NUMBER);
    /* Initialize Rx Descriptors list: Chain Mode  */
    (void)ETH_DMA_RxDescListInit(&EthHandle, EthDmaRxDscrTab, &EthRxBuff[0][0],
ETH_RXBUF_NUMBER);

    /* set MAC hardware address length */
    netif->hwaddr_len = (u8_t)ETH_HWADDR_LEN;

    /* set MAC hardware address */
    netif->hwaddr[0] = (EthHandle.stcCommInit).au8MACAddr[0];
    netif->hwaddr[1] = (EthHandle.stcCommInit).au8MACAddr[1];
    netif->hwaddr[2] = (EthHandle.stcCommInit).au8MACAddr[2];
    netif->hwaddr[3] = (EthHandle.stcCommInit).au8MACAddr[3];
    netif->hwaddr[4] = (EthHandle.stcCommInit).au8MACAddr[4];
    netif->hwaddr[5] = (EthHandle.stcCommInit).au8MACAddr[5];

    /* maximum transfer unit */
    netif->mtu = 1500U;

    /* device capabilities */
    /* don't set NETIF_FLAG_ETHARP if this device is not an ethernet one */
    netif->flags |= NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP;

    /* Enable MAC and DMA transmission and reception */
    (void)ETH_Start();

    /* Configure PHY LED mode */
    u16RegVal = PHY_PAGE_ADDR_7;
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_PSR, u16RegVal);
    (void)ETH_PHY_ReadRegister(&EthHandle, PHY_P7_IWLFR, &u16RegVal);
    MODIFY_REG16(u16RegVal, PHY_LED_SELECT, PHY_LED_SELECT_10);
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_P7_IWLFR, u16RegVal);
    u16RegVal = PHY_PAGE_ADDR_0;
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_PSR, u16RegVal);

#ifdef ETH_INTERFACE_RMII
    /* Disable Power Saving Mode */
    (void)ETH_PHY_ReadRegister(&EthHandle, PHY_PSMR, &u16RegVal);
    CLEAR_REG16_BIT(u16RegVal, PHY_EN_PWR_SAVE);
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_PSMR, u16RegVal);

    /* Configure PHY to generate an interrupt when Eth Link state changes */
    u16RegVal = PHY_PAGE_ADDR_7;
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_PSR, u16RegVal);
    /* Enable Interrupt on change of link status */
    (void)ETH_PHY_ReadRegister(&EthHandle, PHY_P7_IWLFR, &u16RegVal);
    SET_REG16_BIT(u16RegVal, PHY_INT_LINK_CHANGE);
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_P7_IWLFR, u16RegVal);
    u16RegVal = PHY_PAGE_ADDR_0;
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_PSR, u16RegVal);
#endif
}

/**
 * @brief This function should do the actual transmission of the packet. The packet is
 * contained in the pbuf that is passed to the function. This pbuf
 * might be chained.
 *
 * @param netif the lwip network interface structure for this ethernetif
 * @param p the MAC packet to send (e.g. IP packet including MAC addresses and type)
 * @return ERR_OK if the packet could be sent
 *         an err_t value if the packet couldn't be sent
 *
 * @note Returning ERR_MEM here if a DMA queue of your MAC is full can lead to
 *       strange results. You might consider waiting for space in the DMA queue
 *       to become available since the stack doesn't retry to send a packet
 *       dropped because of memory failure (except for the TCP timers).
 */
static err_t low_level_output(struct netif *netif, struct pbuf *p)
{
    err_t errval;
    struct pbuf *q;
    uint8_t *txBuffer;
    __IO stc_eth_dma_desc_t *DmaTxDesc;
    uint32_t byteCnt;
    uint32_t frameLength = 0UL;
    uint32_t bufferOffset;
    uint32_t payloadOffset;

    DmaTxDesc = EthHandle.stcTxDesc;
    txBuffer = (uint8_t *)((EthHandle.stcTxDesc)->u32Buffer1Addr);
    bufferOffset = 0UL;

    /* Copy frame from pbufs to driver buffers */
    for (q = p; q != NULL; q = q->next)
    {
        /* If this buffer isn't available, goto error */
        if (0UL != (DmaTxDesc->u32ControlStatus & ETH_DMATXDESC_OWN))
        {
            errval = (err_t)ERR_USE;
            goto error;
        }

        /* Get bytes in current lwIP buffer */
        byteCnt = q->len;
        payloadOffset = 0UL;
        /* Check if the length of data to copy is bigger than Tx buffer size */
        while ((byteCnt + bufferOffset) > ETH_TXBUF_SIZE)
        {
            /* Copy data to Tx buffer*/
            (void)memcpy((uint8_t *)&(txBuffer[bufferOffset]), (uint8_t *)&(((uint8_t *)q->payload)[payloadOffset]), (ETH_TXBUF_SIZE - bufferOffset));
            /* Point to next descriptor */
            DmaTxDesc = (stc_eth_dma_desc_t *)(DmaTxDesc->u32Buffer2NextDescAddr);
            /* Check if the buffer is available */
            if (0UL != (DmaTxDesc->u32ControlStatus & ETH_DMATXDESC_OWN))
            {
                errval = (err_t)ERR_USE;
                goto error;
            }

            txBuffer = (uint8_t *)(DmaTxDesc->u32Buffer1Addr);
            byteCnt = byteCnt - (ETH_TXBUF_SIZE - bufferOffset);
            payloadOffset = payloadOffset + (ETH_TXBUF_SIZE - bufferOffset);
            frameLength = frameLength + (ETH_TXBUF_SIZE - bufferOffset);
            bufferOffset = 0UL;
        }

        /* Copy the remaining bytes */
        (void)memcpy((uint8_t *)&(txBuffer[bufferOffset]), (uint8_t *)&(((uint8_t *)q->payload)[payloadOffset]), byteCnt);
        bufferOffset = bufferOffset + byteCnt;
        frameLength = frameLength + byteCnt;
    }
    /* Prepare transmit descriptors to give to DMA */
    (void)ETH_DMA_SetTransmitFrame(&EthHandle, frameLength);
    errval = (err_t)ERR_OK;

error:
    /* When Transmit Underflow flag is set, clear it and issue a Transmit Poll Demand to resume transmission */
    if (0UL != (READ_REG32_BIT(M4_ETH->DMA_DMASTSR, ETH_DMA_FLAG_UNS)))
    {
        /* Clear DMA UNS flag */
        WRITE_REG32(M4_ETH->DMA_DMASTSR, ETH_DMA_FLAG_UNS);
        /* Resume DMA transmission */
        WRITE_REG32(M4_ETH->DMA_TXPOLLR, 0UL);
    }

    return errval;
}

/**
 * @brief Should allocate a pbuf and transfer the bytes of the incoming
 * packet from the interface into the pbuf.
 *
 * @param netif the lwip network interface structure for this ethernetif
 * @return a pbuf filled with the received packet (including MAC header)
 *         NULL on memory error
 */
static struct pbuf *low_level_input(struct netif *netif)
{
    struct pbuf *p = NULL;
    struct pbuf *q;
    uint32_t len;
    uint8_t *rxBuffer;
    __IO stc_eth_dma_desc_t *DmaRxDesc;
    uint32_t byteCnt;
    uint32_t bufferOffset;
    uint32_t payloadOffset;
    uint32_t i;

    /* Get received frame */
    if (Ok != ETH_DMA_GetReceiveFrame(&EthHandle))
    {
        return NULL;
    }

    /* Obtain the size of the packet */
    len = (EthHandle.stcRxFrame).u32Length;
    rxBuffer = (uint8_t *)(EthHandle.stcRxFrame).u32Buffer;
    if (len > 0UL)
    {
        /* Allocate a pbuf chain of pbufs from the Lwip buffer pool */
        p = pbuf_alloc(PBUF_RAW, (uint16_t)len, PBUF_POOL);
    }

    if (p != NULL)
    {
        DmaRxDesc = (EthHandle.stcRxFrame).pstcFSDesc;
        bufferOffset = 0UL;
        for (q = p; q != NULL; q = q->next)
        {
            byteCnt = q->len;
            payloadOffset = 0UL;

            /* Check if the length of bytes to copy in current pbuf is bigger than Rx buffer size */
            while ((byteCnt + bufferOffset) > ETH_RXBUF_SIZE)
            {
                /* Copy data to pbuf */
                (void)memcpy((uint8_t *)&(((uint8_t *)q->payload)[payloadOffset]), (uint8_t
*)&(rxBuffer[bufferOffset]), (ETH_RXBUF_SIZE - bufferOffset));
                /* Point to next descriptor */
                DmaRxDesc = (stc_eth_dma_desc_t *)(DmaRxDesc->u32Buffer2NextDescAddr);
                rxBuffer = (uint8_t *)(DmaRxDesc->u32Buffer1Addr);
                byteCnt = byteCnt - (ETH_RXBUF_SIZE - bufferOffset);
                payloadOffset = payloadOffset + (ETH_RXBUF_SIZE - bufferOffset);
                bufferOffset = 0UL;
            }

            /* Copy remaining data in pbuf */
            (void)memcpy((uint8_t *)&(((uint8_t *)q->payload)[payloadOffset]), (uint8_t
*)&(rxBuffer[bufferOffset]), byteCnt);
            bufferOffset = bufferOffset + byteCnt;
        }
    }

    /* Release descriptors to DMA */
    /* Point to first descriptor */
    DmaRxDesc = (EthHandle.stcRxFrame).pstcFSDesc;
    for (i = 0UL; i < (EthHandle.stcRxFrame).u32SegCount; i++)
    {
        DmaRxDesc->u32ControlStatus |= ETH_DMARXDESC_OWN;
        DmaRxDesc = (stc_eth_dma_desc_t *)(DmaRxDesc->u32Buffer2NextDescAddr);
    }
    /* Clear Segment_Count */
    (EthHandle.stcRxFrame).u32SegCount = 0UL;

    /* When Rx Buffer unavailable flag is set, clear it and resume reception */
    if (0UL != (READ_REG32_BIT(M4_ETH->DMA_DMASTSR, ETH_DMA_FLAG_RUS)))
    {
        /* Clear DMA RUS flag */
        WRITE_REG32(M4_ETH->DMA_DMASTSR, ETH_DMA_FLAG_RUS);
        /* Resume DMA reception */
        WRITE_REG32(M4_ETH->DMA_RXPOLLR, 0UL);
    }

    return p;
}

/**
 * @brief This function should be called when a packet is ready to be read
 * from the interface. It uses the function low_level_input() that
 * should handle the actual reception of bytes from the network
 * interface. Then the type of the received packet is determined and
 * the appropriate input function is called.
 *
 * @param netif the lwip network interface structure for this ethernetif
 */
void ethernetif_input(struct netif *netif)
{
    err_t err;
    struct pbuf *p;

    /* Move received packet into a new pbuf */
    p = low_level_input(netif);
    /* No packet could be read, silently ignore this */
    if (p == NULL)
    {
        return;
    }

    /* Entry point to the LwIP stack */
    err = netif->input(p, netif);
    if (err != (err_t)ERR_OK)
    {
        LWIP_DEBUGF(NETIF_DEBUG, ("ethernetif_input: IP input error\n"));
        (void)pbuf_free(p);
    }
}

/**
 * @brief Should be called at the beginning of the program to set up the
 * network interface. It calls the function low_level_init() to do the
 * actual setup of the hardware.
 *
 * This function should be passed as a parameter to netif_add().
 *
 * @param netif the lwip network interface structure for this ethernetif
 * @return ERR_OK if the IF is initialized
 *         ERR_MEM if private data couldn't be allocated
 *         any other err_t on error
 */
err_t ethernetif_init(struct netif *netif)
{
#if LWIP_NETIF_HOSTNAME
    /* Initialize interface hostname */
    netif->hostname = "lwip";
#endif /* LWIP_NETIF_HOSTNAME */

    netif->name[0] = IFNAME0;
    netif->name[1] = IFNAME1;
    /* We directly use etharp_output() here to save a function call.
     * You can instead declare your own function an call etharp_output()
     * from it if you have to do some checks before sending (e.g. if link
     * is available...) */
    netif->output = &etharp_output;
    netif->linkoutput = &low_level_output;

    /* initialize the hardware */
    low_level_init(netif);

    return (err_t)ERR_OK;
}

/**
 * @brief  Returns the current time in milliseconds when LWIP_TIMERS == 1 and NO_SYS
== 1
 * @param  None
 * @retval Current Time value
 */
u32_t sys_now(void)
{
    return SysTick_GetTick();
}

/**
 * @brief  Check the netif link status.
 * @param  netif the network interface
 * @retval None
 */
void EthernetIF_CheckLink(struct netif *netif)
{
    static uint8_t u8PreStatus = 0U;
    uint16_t u16RegVal = 0U;

    /* Read PHY_BSR */
    (void)ETH_PHY_ReadRegister(&EthHandle, PHY_BSR, &u16RegVal);
    /* Check whether the link is up or down*/
    if ((0x0000U != u16RegVal) && (0xFFFFU != u16RegVal))
    {
        if ((0U != (u16RegVal & PHY_LINK_STATUS)) && (0U == u8PreStatus))
        {
            netif_set_link_up(netif);
            u8PreStatus = 1U;
        }

        if ((0U == (u16RegVal & PHY_LINK_STATUS)) && (1U == u8PreStatus))
        {
            netif_set_link_down(netif);
            u8PreStatus = 0U;
        }
    }
}

/**
 * @brief  Update the netif link status.
 * @param  netif the network interface
 * @retval None
 */
void EthernetIF_UpdateLink(struct netif *netif)
{
    uint16_t u16RegVal;

    u16RegVal = PHY_PAGE_ADDR_0;
    (void)ETH_PHY_WriteRegister(&EthHandle, PHY_PSR, u16RegVal);
    /* Read PHY_IISDR */
    (void)ETH_PHY_ReadRegister(&EthHandle, PHY_IISDR, &u16RegVal);
    /* Check whether the link interrupt has occurred or not */
    if (0U != (u16RegVal & PHY_FLAG_LINK_STATUS_CHANGE))
    {
        /* Read PHY_BSR */
        (void)ETH_PHY_ReadRegister(&EthHandle, PHY_BSR, &u16RegVal);
        if ((0x0000U != u16RegVal) && (0xFFFFU != u16RegVal))
        {
            if (!netif_is_link_up(netif))
            {
                /* Wait until the auto-negotiation will be completed */
                SysTick_Delay(2U);
                (void)ETH_PHY_ReadRegister(&EthHandle, PHY_BSR, &u16RegVal);
            }

            /* Check whether the link is up or down*/
            if (0U != (u16RegVal & PHY_LINK_STATUS))
            {
                netif_set_link_up(netif);
            }
            else
            {
                netif_set_link_down(netif);
            }
        }
    }
}
```

```
/**
 * @brief  Link callback function
 * @note   This function is called on change of link status to update low level
 *         driver configuration.
 * @param  netif The network interface
 * @retval None
 */
void EthernetIF_LinkCallback(struct netif *netif)
{
    __IO uint32_t tickStart = 0UL;
    uint16_t u16RegVal = 0U;
    __IO en_result_t negoResult = Error;

    if (netif_is_link_up(netif))
    {
        /* Restart the auto-negotiation */
        if (ETH_AUTO_NEGOTIATION_DISABLE !=
(EthHandle.stcCommInit).u16AutoNegotiation)
        {
            /* Enable Auto-Negotiation */
            (void)ETH_PHY_ReadRegister(&EthHandle, PHY_BCR, &u16RegVal);
            u16RegVal |= PHY_AUTONEGOTIATION;
            (void)ETH_PHY_WriteRegister(&EthHandle, PHY_BCR, u16RegVal);

            /* Wait until the auto-negotiation will be completed */
            tickStart = SysTick_GetTick();
            do
            {
                (void)ETH_PHY_ReadRegister(&EthHandle, PHY_BSR, &u16RegVal);
                if (PHY_AUTONEGO_COMPLETE == (u16RegVal &
PHY_AUTONEGO_COMPLETE))
                {
                    break;
                }
                /* Check for the Timeout (3s) */
            } while ((SysTick_GetTick() - tickStart) <= 3000U);

            if (PHY_AUTONEGO_COMPLETE == (u16RegVal &
PHY_AUTONEGO_COMPLETE))
            {
                negoResult = Ok;
                /* Configure ETH duplex mode according to the result of automatic negotiation */
                if (0U != (u16RegVal & (PHY_100BASE_TX_FD | PHY_10BASE_T_FD)))
                {
                    (EthHandle.stcCommInit).u32DuplexMode =
ETH_MAC_MODE_FULLDUPLEX;
                }
                else
                {
                    (EthHandle.stcCommInit).u32DuplexMode =
ETH_MAC_MODE_HALFDUPLEX;
                }

                /* Configure ETH speed according to the result of automatic negotiation */
                if (0U != (u16RegVal & (PHY_100BASE_TX_FD | PHY_100BASE_TX_HD)))
        {
          (EthHandle.stcCommInit).u32Speed = ETH_MAC_SPEED_100M;
        }
        else
        {
          (EthHandle.stcCommInit).u32Speed = ETH_MAC_SPEED_10M;
        }
      }
    }

    /* AutoNegotiation disable or failed*/
    if (Error == negoResult)
    {
      (void)ETH_PHY_ReadRegister(&EthHandle, PHY_BCR, &u16RegVal);
      u16RegVal &= ~0x2100;
      /* Set MAC Speed and Duplex Mode to PHY */
      (void)ETH_PHY_WriteRegister(&EthHandle, PHY_BCR,
                    ((uint16_t)((EthHandle.stcCommInit).u32DuplexMode >> 3U) |
                     (uint16_t)((EthHandle.stcCommInit).u32Speed >> 1U) | u16RegVal));
    }

    /* ETH MAC Re-Configuration */
    ETH_MAC_SetDuplexSpeed((EthHandle.stcCommInit).u32DuplexMode,
(EthHandle.stcCommInit).u32Speed);
    /* Restart MAC interface */
    (void)ETH_Start();
  }
  else
  {
    /* Stop MAC interface */
    (void)ETH_Stop();
  }

  /* Notify link status change */
  EthernetIF_NotifyLinkChange(netif);
}

/**
 * @brief  Notify link status change.
 * @param  netif the network interface
 * @retval None
 */
__WEAKDEF void EthernetIF_NotifyLinkChange(struct netif *netif)
{
  /* This is function could be implemented in user file
     when the callback is needed */
}
```

5.3.2 修改 ethernetif.h 文件

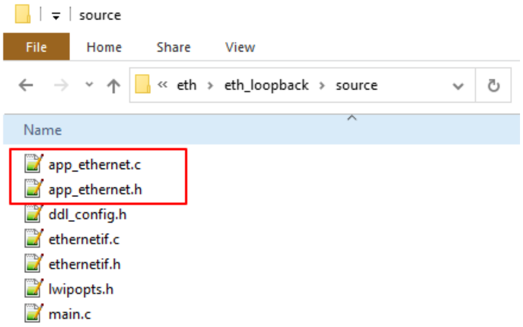
```
#ifndef __ETHERNETIF_H__
#define __ETHERNETIF_H__

/* C binding of definitions if building with C++ compiler */
#ifdef __cplusplus
extern "C"
{
#endif

#include "lwip/err.h"
#include "lwip/netif.h"

/* Ethernet PHY interface */
// #define ETH_INTERFACE_RMII

/* Extended PHY Registers */
#define PHY_PSMR                        (0x18U)   /*!< Power Saving Mode Register */
#define PHY_IISDR                       (0x1EU)   /*!< Interrupt Indicators and SNR Display
Register */
#define PHY_PSR                         (0x1FU)   /*!< Page Select Register */

#define PHY_P7_RMSR                     (0x10U)   /*!< RMII Mode Setting Register */
#define PHY_P7_IWLFR                    (0x13U)   /*!< Interrupt, WOL Enable, and LED
Function Registers  */

/* The following parameters will return to default values after a software reset */
#define PHY_EN_PWR_SAVE                       (0x8000U)   /*!< Enable Power Saving Mode /

#define PHY_FLAG_AUTO_NEGO_ERROR              (0x8000U)   /*!< Auto-Negotiation
Error Interrupt Flag  */
#define PHY_FLAG_SPEED_MODE_CHANGE            (0x4000U)   /*!< Speed Mode
Change Interrupt Flag */
#define PHY_FLAG_DUPLEX_MODE_CHANGE           (0x2000U)   /*!< Duplex Mode
Change Interrupt Flag */
#define PHY_FLAG_LINK_STATUS_CHANGE           (0x0800U)   /*!< Link Status Change
Interrupt Flag */

#define PHY_PAGE_ADDR_0                       (0x0000U)   /*!< Page Address 0 (default) */
#define PHY_PAGE_ADDR_7                       (0x0007U)   /*!< Page Address 7 */

#define PHY_RMII_CLK_DIR                      (0x1000U) /*!< TXC direction in RMII Mode */
#define PHY_RMII_MODE                       (0x0008U)   /*!< RMII Mode or MII Mode */
#define PHY_RMII_RXDV_CRSDV                       (0x0004U) /*!< CRS_DV or RXDV select */

#define PHY_INT_LINK_CHANGE                       (0x2000U)   /*!< Link Change Interrupt
Mask */
#define PHY_INT_DUPLEX_CHANGE                       (0x1000U)   /*!< Duplex Change Interrupt
Mask */
#define PHY_INT_AUTO_NEGO_ERROR                    (0x0800U)   /*!< Auto-Negotiation Error
Interrupt Mask */
#define PHY_LED_WOL_SELECT                    (0x0400U)   /*!< LED and Wake-On-LAN
Function Selection */
#define PHY_LED_SELECT                        (0x0030U)   /*!< Traditional LED Function
Selection.                 */
#define PHY_LED_SELECT_00                     (0x0000U)   /*!< LED0: ACT(all)      LED1:
LINK(100)           */
#define PHY_LED_SELECT_01                     (0x0010U)   /*!< LED0: LINK(ALL)/ACT(all)
LED1: LINK(100)           */
#define PHY_LED_SELECT_10                     (0x0020U)   /*!< LED0: LINK(10)/ACT(all)
LED1: LINK(100)           */
#define PHY_LED_SELECT_11                     (0x0030U)   /*!< LED0: LINK(10)/ACT(10)
LED1: LINK(100)/ACT(100)  */
#define PHY_EN_10M_LED_FUNC                   (0x0001U)   /*!< Enable 10M LPI LED
Function      */

err_t ethernetif_init(struct netif *netif);
void ethernetif_input(struct netif *netif);

void EthernetIF_CheckLink(struct netif *netif);
void EthernetIF_UpdateLink(struct netif *netif);
void EthernetIF_LinkCallback(struct netif *netif);
void EthernetIF_NotifyLinkChange(struct netif *netif);

#ifdef __cplusplus
}
#endif

#endif /* __ETHERNETIF_H__ */
```

5.4 Http 服务器功能测试

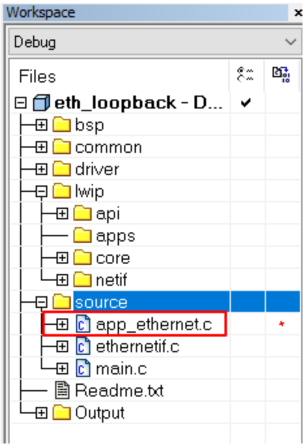
网卡驱动移植完成后，为了验证 LwIP 能否基于 ETH 正常运行起来，所以需要修改 lwipopts.h 的配置和增加协议栈初始化相关代码，并在此基础上搭建一个 Http 服务器用来测试 LwIP，故需要进行如下的修改操作。

5.4.1 创建 app_ethernet 文件并添加到工程

1) 在“..\eth\eth_loopback\source”文件夹下创建 app_ethernet.c 和 app_ethernet.h 文件，如下图：

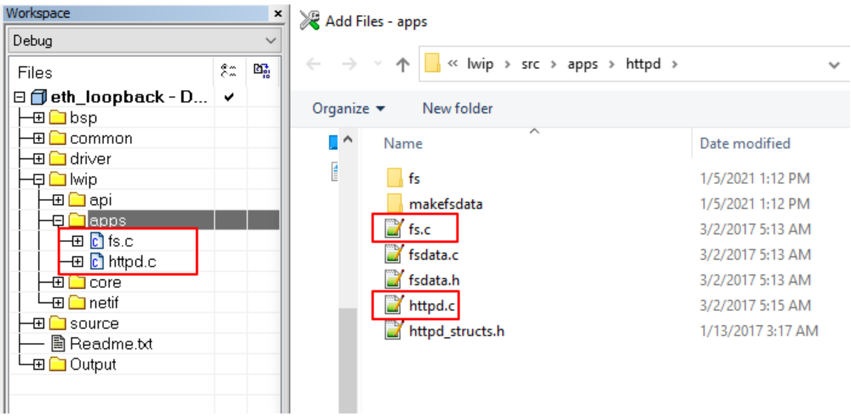


2) 添加 app_ethernet.c 到工程中，如下图：



5.4.2 添加 HTTP 服务器文件到工程

样例演示的是一个 http 服务器，所以需要将“..\lwip\src\apps\httpd”目录下的 fs.c 和 httpd.c 添加到工程中，添加后如下图：



5.4.3 编辑 app_ethernet.c 文件

文件中程序如下：

```
#include "hc32_ddl.h"
#include "app_ethernet.h"
#include "ethernetif.h"

#ifndef ETH_INTERFACE_RMII
static uint32_t u32LinkTimer = 0UL;
#endif

/**
 * @brief  Notify network connect status of Ethernet.
 * @param  [in] netif                   Pointer to a struct netif structure
 * @retval None
 */
void Ethernet_NotifyConnStatus(struct netif *netif)
{
    if (netif_is_up(netif))
    {
        /* Turn On LED BULE to indicate ETH and LwIP init success*/
        BSP_LED_On(LED_BLUE);
    }
    else
    {
        /* Turn On LED RED to indicate ETH and LwIP init error */
        BSP_LED_On(LED_RED);
    }
}

/**
 * @brief  Notify link status change.
 * @param  [in] netif                   Pointer to a struct netif structure
 * @retval None
 */
void EthernetIF_NotifyLinkChange(struct netif *netif)
{
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;

    if (netif_is_link_up(netif))
    {
        IP_ADDR4(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
        IP_ADDR4(&netmask, NETMASK_ADDR0, NETMASK_ADDR1,
NETMASK_ADDR2, NETMASK_ADDR3);
        IP_ADDR4(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);
        netif_set_addr(netif, &ipaddr, &netmask, &gw);
        /* When the netif is fully configured this function must be called. */
        netif_set_up(netif);

        BSP_LED_Off(LED_RED);
        BSP_LED_On(LED_BLUE);
    }
    else
    {
        /*  When the netif link is down this function must be called. */
        netif_set_down(netif);

        BSP_LED_Off(LED_BLUE);
        BSP_LED_On(LED_RED);
    }
}

/**
 * @brief  LwIP periodic handle
 * @param  [in] netif                   Pointer to a struct netif structure
 * @retval None
 */
void LwIP_PeriodicHandle(struct netif *netif)
{
#ifndef ETH_INTERFACE_RMII
    uint32_t curTick;

    curTick = SysTick_GetTick();
    /* Check link status periodically */
    if ((curTick - u32LinkTimer) >= LINK_TIMER_INTERVAL)
    {
        u32LinkTimer = curTick;
        EthernetIF_CheckLink(netif);
    }
#endif /* ETH_INTERFACE_RMII */
}
```

5.4.4 编辑 app_ethernet.h 文件

文件中程序如下：

```
#ifndef __APP_ETHERNET_H__
#define __APP_ETHERNET_H__

/* C binding of definitions if building with C++ compiler */
#ifdef __cplusplus
extern "C"
{
#endif

#include "lwip/netif.h"

/* Number of milliseconds when to check for link status from PHY */
#ifndef LINK_TIMER_INTERVAL
   #define LINK_TIMER_INTERVAL              (1000U)
#endif

/* Static IP Address */
#define IP_ADDR0                        (192U)
#define IP_ADDR1                        (168U)
#define IP_ADDR2                        (1U)
#define IP_ADDR3                        (20U)

/* Static Netmask */
#define NETMASK_ADDR0                   (255U)
#define NETMASK_ADDR1                   (255U)
#define NETMASK_ADDR2                   (255U)
#define NETMASK_ADDR3                   (0U)

/* Static Gateway Address*/
#define GW_ADDR0                        (192U)
#define GW_ADDR1                        (168U)
#define GW_ADDR2                        (1U)
#define GW_ADDR3                        (1U)

void Ethernet_NotifyConnStatus(struct netif *netif);
void LwIP_PeriodicHandle(struct netif *netif);

#ifdef __cplusplus
}
#endif

#endif /* __APP_ETHERNET_H__ */
```

5.4.5 修改 lwipopts.h 文件

```
#ifndef __LWIPOPTS_H__
#define __LWIPOPTS_H__

/**
 * NO_SYS==1: Provides VERY minimal functionality. Otherwise,
 * use lwIP facilities.
 */
#define NO_SYS                  1

/**
 * SYS_LIGHTWEIGHT_PROT==1: if you want inter-task protection for certain
 * critical regions during buffer allocation, deallocation and memory
 * allocation and deallocation.
 */
#define SYS_LIGHTWEIGHT_PROT    0

/* ---------- Memory options ---------- */
/* MEM_ALIGNMENT: should be set to the alignment of the CPU for which
   lwIP is compiled. 4 byte alignment -> define MEM_ALIGNMENT to 4, 2
   byte alignment -> define MEM_ALIGNMENT to 2. */
#define MEM_ALIGNMENT           4

/* MEM_SIZE: the size of the heap memory. If the application will send
a lot of data that needs to be copied, this should be set high. */
#define MEM_SIZE                (10*1024)

/* MEMP_NUM_PBUF: the number of memp struct pbufs. If the application
   sends a lot of data out of ROM (or other static memory), this
   should be set high. */
#define MEMP_NUM_PBUF           10
/* MEMP_NUM_UDP_PCB: the number of UDP protocol control blocks. One
   per active UDP "connection". */
#define MEMP_NUM_UDP_PCB        6
/* MEMP_NUM_TCP_PCB: the number of simulatenously active TCP
   connections. */
#define MEMP_NUM_TCP_PCB        10
/* MEMP_NUM_TCP_PCB_LISTEN: the number of listening TCP
   connections. */
#define MEMP_NUM_TCP_PCB_LISTEN 6
/* MEMP_NUM_TCP_SEG: the number of simultaneously queued TCP
   segments. */
#define MEMP_NUM_TCP_SEG        8
/* MEMP_NUM_SYS_TIMEOUT: the number of simulateously active
   timeouts. */
#define MEMP_NUM_SYS_TIMEOUT    10


/* ---------- Pbuf options ---------- */
/* PBUF_POOL_SIZE: the number of buffers in the pbuf pool. */
#define PBUF_POOL_SIZE          8

/* PBUF_POOL_BUFSIZE: the size of each pbuf in the pbuf pool. */
```

```
#define PBUF_POOL_BUFSIZE       1524


/* ---------- TCP options ---------- */
#define LWIP_TCP                1
#define TCP_TTL                 255

/* Controls if TCP should queue segments that arrive out of
   order. Define to 0 if your device is low on memory. */
#define TCP_QUEUE_OOSEQ         0

/* TCP Maximum segment size. */
#define TCP_MSS                 (1500 - 40)  /* TCP_MSS = (Ethernet MTU - IP header size - TCP
header size) */

/* TCP sender buffer space (bytes). */
#define TCP_SND_BUF             (4*TCP_MSS)

/*  TCP_SND_QUEUELEN: TCP sender buffer space (pbufs). This must be at least
  as much as (2 * TCP_SND_BUF/TCP_MSS) for things to work. */

#define TCP_SND_QUEUELEN        (2* TCP_SND_BUF/TCP_MSS)

/* TCP receive window. */
#define TCP_WND                 (2*TCP_MSS)

/* ---------- IPv4 options ---------- */
#define LWIP_IPV4               1

/* ---------- ICMP options ---------- */
#define LWIP_ICMP               1


/* ---------- DHCP options ---------- */
#define LWIP_DHCP               1


/* ---------- UDP options ---------- */
#define LWIP_UDP                1
#define UDP_TTL                 255


/* ---------- Statistics options ---------- */
#define LWIP_STATS 0

/* ---------- link callback options ---------- */
/* LWIP_NETIF_LINK_CALLBACK==1: Support a callback function from an interface
 * whenever the link changes (i.e., link down)
 */
#define LWIP_NETIF_LINK_CALLBACK        1


/* ---------- Checksum options ---------- */
/*
The chip allows computing and verifying the IP, UDP, TCP and ICMP checksums by hardware:
 - To use this feature let the following define uncommented.
 - To disable it and process by CPU comment the checksum.
*/
#define CHECKSUM_BY_HARDWARE


#ifdef CHECKSUM_BY_HARDWARE
  /* CHECKSUM_GEN_IP==0: Generate checksums by hardware for outgoing IP packets.*/
  #define CHECKSUM_GEN_IP                 0
  /* CHECKSUM_GEN_UDP==0: Generate checksums by hardware for outgoing UDP
packets.*/
  #define CHECKSUM_GEN_UDP                0
  /* CHECKSUM_GEN_TCP==0: Generate checksums by hardware for outgoing TCP
packets.*/
  #define CHECKSUM_GEN_TCP                0
  /* CHECKSUM_CHECK_IP==0: Check checksums by hardware for incoming IP packets.*/
  #define CHECKSUM_CHECK_IP               0
  /* CHECKSUM_CHECK_UDP==0: Check checksums by hardware for incoming UDP
packets.*/
  #define CHECKSUM_CHECK_UDP              0
  /* CHECKSUM_CHECK_TCP==0: Check checksums by hardware for incoming TCP
packets.*/
  #define CHECKSUM_CHECK_TCP              0
  /* CHECKSUM_CHECK_ICMP==0: Check checksums by hardware for incoming ICMP
packets.*/
  #define CHECKSUM_GEN_ICMP               0
#else
  /* CHECKSUM_GEN_IP==1: Generate checksums in software for outgoing IP packets.*/
  #define CHECKSUM_GEN_IP                 1
  /* CHECKSUM_GEN_UDP==1: Generate checksums in software for outgoing UDP
packets.*/
  #define CHECKSUM_GEN_UDP                1
  /* CHECKSUM_GEN_TCP==1: Generate checksums in software for outgoing TCP
packets.*/
  #define CHECKSUM_GEN_TCP                1
  /* CHECKSUM_CHECK_IP==1: Check checksums in software for incoming IP packets.*/
  #define CHECKSUM_CHECK_IP               1
  /* CHECKSUM_CHECK_UDP==1: Check checksums in software for incoming UDP
packets.*/
  #define CHECKSUM_CHECK_UDP              1
  /* CHECKSUM_CHECK_TCP==1: Check checksums in software for incoming TCP
packets.*/
  #define CHECKSUM_CHECK_TCP              1
  /* CHECKSUM_CHECK_ICMP==1: Check checksums by hardware for incoming ICMP
packets.*/
  #define CHECKSUM_GEN_ICMP               1
#endif


/* ---------- Sequential layer options ---------- */
/**
 * LWIP_NETCONN==1: Enable Netconn API (require to use api_lib.c)
 */
#define LWIP_NETCONN                    0
```

```
/* ---------- Socket options ---------- */
/**
 * LWIP_SOCKET==1: Enable Socket API (require to use sockets.c)
 */
#define LWIP_SOCKET                     0

#endif /* __LWIPOPTS_H__ */
```

5.4.6 修改 main.c 文件

```
#include "hc32_ddl.h"
#include "lwip/init.h"
#include "lwip/netif.h"
#include "lwip/timeouts.h"
#include "netif/etharp.h"
#include "ethernetif.h"
#include "app_ethernet.h"
#include "lwip/debug.h"
#include "lwip/tcp.h"
#include "lwip/apps/httpd.h"

#ifdef ETH_INTERFACE_RMII
    /* RMII_INTB */
    #define ETH_RMII_INTB_PORT                  (GPIO_PORT_B)
    #define ETH_RMII_INTB_PIN                   (GPIO_PIN_00)
    #define ETH_RMII_INTB_EXINT_CH              (EXINT_CH00)
    #define ETH_RMII_INTB_INT_SRC               (INT_PORT_EIRQ0)
    #define ETH_RMII_INTB_IRQn                  (Int037_IRQn)
#endif

static struct netif gnetif;

/**
 * @brief  SysTick interrupt callback function.
 * @param  None
 * @retval None
 */
void SysTick_IrqHandler(void)
{
    SysTick_IncTick();
}

/**
 * @brief  MCU Peripheral registers write unprotected.
 * @param  None
 * @retval None
 * @note Comment/uncomment each API depending on APP requires.
 */
static void Peripheral_WE(void)
{
    /* Unlock GPIO register: PSPCR, PCCR, PINAER, PCRxy, PFSRxy */
    GPIO_Unlock();
    /* Unlock PWC register: FCG0 */
    PWC_FCG0_Unlock();
    /* Unlock PWC, CLK, PVD registers, @ref PWC_REG_Write_Unlock_Code for details */
    PWC_Unlock(PWC_UNLOCK_CODE_0 | PWC_UNLOCK_CODE_1);
    /* Unlock SRAM register: WTCR */
    SRAM_WTCR_Unlock();
    /* Unlock SRAM register: CKCR */
    // SRAM_CKCR_Unlock();
    /* Unlock all EFM registers */
    EFM_Unlock();
    /* Unlock EFM register: FWMC */
    // EFM_FWMC_Unlock();
    /* Unlock EFM OTP write protect registers */
    // EFM_OTP_WP_Unlock();
    /* Unlock all MPU registers */
    // MPU_Unlock();
}

/**
 * @brief  MCU Peripheral registers write protected.
 * @param  None
 * @retval None
 * @note Comment/uncomment each API depending on APP requires.
 */
static void Peripheral_WP(void)
{
    /* Lock GPIO register: PSPCR, PCCR, PINAER, PCRxy, PFSRxy */
    // GPIO_Lock();
    /* Lock PWC register: FCG0 */
    PWC_FCG0_Lock();
    /* Lock PWC, CLK, PVD registers, @ref PWC_REG_Write_Unlock_Code for details */
    // PWC_Lock(PWC_UNLOCK_CODE_0 | PWC_UNLOCK_CODE_1 |
PWC_UNLOCK_CODE_2);
    /* Lock SRAM register: WTCR */
    SRAM_WTCR_Lock();
    /* Lock SRAM register: CKCR */
    // SRAM_CKCR_Lock();
    /* Lock EFM OTP write protect registers */
    // EFM_OTP_WP_Lock();
    /* Lock EFM register: FWMC */
    // EFM_FWMC_Lock();
    /* Lock all EFM registers */
    EFM_Lock();
    /* Lock all MPU registers */
    // MPU_Lock();
}

#ifdef ETH_INTERFACE_RMII
/**
 * @brief  ETH RMII interrupt callback function.
 * @param  None
 * @retval None
 */
static void ETH_RMII_LinkIrqCallback(void)
{
    if (Set == EXINT_GetExIntSrc(ETH_RMII_INTB_EXINT_CH))
    {
        EthernetIF_UpdateLink(&gnetif);
        EXINT_ClrExIntSrc(ETH_RMII_INTB_EXINT_CH);
    }
}

/**
 * @brief  ETH RMII link interrupt configuration.
 * @param  None
 * @retval None
 */
static void ETH_RMII_LinkIntConfig(void)
{
    stc_exint_init_t stcExintInit;
    stc_irq_signin_config_t stcIrqSignConfig;
    stc_gpio_init_t stcGpioInit;

    /* GPIO config */
    (void)GPIO_StructInit(&stcGpioInit);
    stcGpioInit.u16ExInt  = PIN_EXINT_ON;
    stcGpioInit.u16PullUp = PIN_PU_ON;
    (void)GPIO_Init(ETH_RMII_INTB_PORT, ETH_RMII_INTB_PIN, &stcGpioInit);

    /* Exint config */
    (void)EXINT_StructInit(&stcExintInit);
    stcExintInit.u32ExIntCh  = ETH_RMII_INTB_EXINT_CH;
    stcExintInit.u32ExIntLvl = EXINT_TRIGGER_FALLING;
    (void)EXINT_Init(&stcExintInit);

    /* IRQ sign-in */
    stcIrqSignConfig.enIntSrc    = ETH_RMII_INTB_INT_SRC;
    stcIrqSignConfig.enIRQn      = ETH_RMII_INTB_IRQn;
    stcIrqSignConfig.pfnCallback = &ETH_RMII_LinkIrqCallback;
    (void)INTC_IrqSignIn(&stcIrqSignConfig);

    /* NVIC config */
    NVIC_ClearPendingIRQ(stcIrqSignConfig.enIRQn);
    NVIC_SetPriority(stcIrqSignConfig.enIRQn, DDL_IRQ_PRIORITY_DEFAULT);
    NVIC_EnableIRQ(stcIrqSignConfig.enIRQn);
}
#endif /* ETH_INTERFACE_RMII */

/**
 * @brief  Configurate the BSP.
 * @param  None
 * @retval None
 */
static void BSP_Config(void)
{
    /* Configure BSP */
    BSP_IO_Init();
    BSP_LED_Init();
    /* Configure UART */
    (void)DDL_PrintfInit();
    /* SysTick configuration */
    (void)SysTick_Init(1000U);
    /* Set Systick Interrupt to the highest priority */
    NVIC_SetPriority(SysTick_IRQn, DDL_IRQ_PRIORITY_00);
    /* Configure link interrupt IO for ETH RMII */
#ifdef ETH_INTERFACE_RMII
    ETH_RMII_LinkIntConfig();
#endif
}

/**
 * @brief  Configurate the network interface
 * @param  None
 * @retval None
 */
static void Netif_Config(void)
{
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;

    IP_ADDR4(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
    IP_ADDR4(&netmask, NETMASK_ADDR0, NETMASK_ADDR1, NETMASK_ADDR2,
NETMASK_ADDR3);
    IP_ADDR4(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);

    /* Add the network interface */
    (void)netif_add(&gnetif, &ipaddr, &netmask, &gw, NULL, &ethernetif_init,
&ethernet_input);
    /*  Registers the default network interface */
    netif_set_default(&gnetif);

    if (netif_is_link_up(&gnetif))
    {
        /* When the netif is fully configured this function must be called */
        netif_set_up(&gnetif);
    }
    else
    {
        /* When the netif link is down this function must be called */
        netif_set_down(&gnetif);
    }
    /* Set the link callback function, this function is called on change of link status*/
    netif_set_link_callback(&gnetif, EthernetIF_LinkCallback);
}

/**
 * @brief  Main function of ETH LWIP_HTTP_Server_RAW.
 * @param  None
 * @retval int32_t return value, if needed
 */
int32_t main(void)
{
    /* Peripheral registers write unprotected */
    Peripheral_WE();
    /* Configure clock */
    BSP_CLK_Init();
    /* Configure the BSP */
    BSP_Config();
    /* Initialize the LwIP stack */
    lwip_init();
    /* Configure the Network interface */
    Netif_Config();
    /* Http webserver Init */
    httpd_init();
    /* Notify user about the network interface config */
    Ethernet_NotifyConnStatus(&gnetif);
    /* Peripheral registers write protected */
    Peripheral_WP();

    for (;;)
    {
        /* Read a received packet from the ETH buffers and send it
           to the lwIP for handling */
        ethernetif_input(&gnetif);

        /* Handle timeouts */
        sys_check_timeouts();
        /* Handle periodic timers */
        LwIP_PeriodicHandle(&gnetif);
    }
}
```

5.4.7 验证功能

1) 使用网线将目标板（EV_F4A0_LQ176_V10）的网口（J26）与 PC 的网络端口连接起来，设置 PC 的对应网卡的 IP 地址信息，如下：

   IP address：192.168.1.120

   Subnet mask: 255.255.255.0

   Default gateway: 192.168.1.1

   Preferred DNS server: 192.168.1.1

2) 打开工程并重新编译；

3) 启动 IDE 的下载和调试功能，关闭 IDE 下载界面；

4) 等待目标板 LED_B 亮起，打开浏览器输入 IP 地址：192.168.1.20 后单击回车键，访问目标；

5) 板的 HTTP 服务器，成功访问并回显 lwip 的简介网页则功能正常。

6 版本信息&联系方式

日期	版本	修改记录
2021/7/27	Rev1.0	初版发布

如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email：mcu@hdsc.com.cn

网址：http://www.hdsc.com.cn/mcu.htm

通信地址：上海市浦东新区中科路 1867 号 A 座 10 层

邮编：201203