Network protocol stack driver porting notes

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Table of contents

This application note describes how to use RT-Thread to configure the hardware.

Network driver, and flexibly use debugging methods to solve problems.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

Most BSPs supported by RT-Thread include Ethernet drivers. However, depending on your hardware, the default code may differ. This article examines the STM32F40x, a relatively well-developed Ethernet driver, and describes the main driver implementations and how to modify them for different hardware types.

1.2 Structure of this paper

This article first introduces common Ethernet chip types and some common terms. It then describes in detail the RT-Thread Ethernet driver architecture, driver interface, and driver porting. It also provides code examples ported on the Zhengdian Atom STM32F4 Explorer development board. Finally, it introduces the driver debugging method.

2 Common Ethernet chip types

There are many types of Ethernet chips, which can be roughly divided into three categories:

- Ethernet chips only have PHY (physical interface transceiver), and require a microcontroller with MAC (Ethernet Media Access Controller) to communicate with the microcontroller through the MII or RMII interface. For example, LAN8720.
- Ethernet chip with MAC and PHY, communicates with MCU via SPI interface. For example, ENC28J60. • Ethernet chip with MAC and PHY, communicates with MCU via SPI interface, and has built-in hardware protocol stack, suitable for Suitable for low-speed microcontrollers, such as W5500.

3 Common noun explanations

MAC: Media Access Control layer, a sublayer below the data link layer in the OSI model.

PHY: PHY refers to the physical layer, the lowest layer of OSI. It generally refers to the chip that interfaces with external signals.

MII: MII (Media Independent Interface), media independent interface, also known as media independent interface, it It is an Ethernet industry standard defined by IEEE-802.3 and supports 10Mbit/s and 100Mbit/s data transmission modes.

RMII: Reduced Media Independent Interface (RMII) is an alternative to the MII interface in the IEEE 802.3u standard. It supports 10Mbit/s and 100Mbit/s data transmission modes. Compared to MII, it has a reduced number of pins.

lwIP: lwIP is a small open source library developed by Adam Dunkels of the Swedish Institute of Computer Science (SICS). TCP/IP The focus of the implementation is to reduce RAM usage while maintaining the main functions of the TCP protocol.

pbuf: A structure used to manage data packets in lwIP.

4 Driver Architecture Diagram

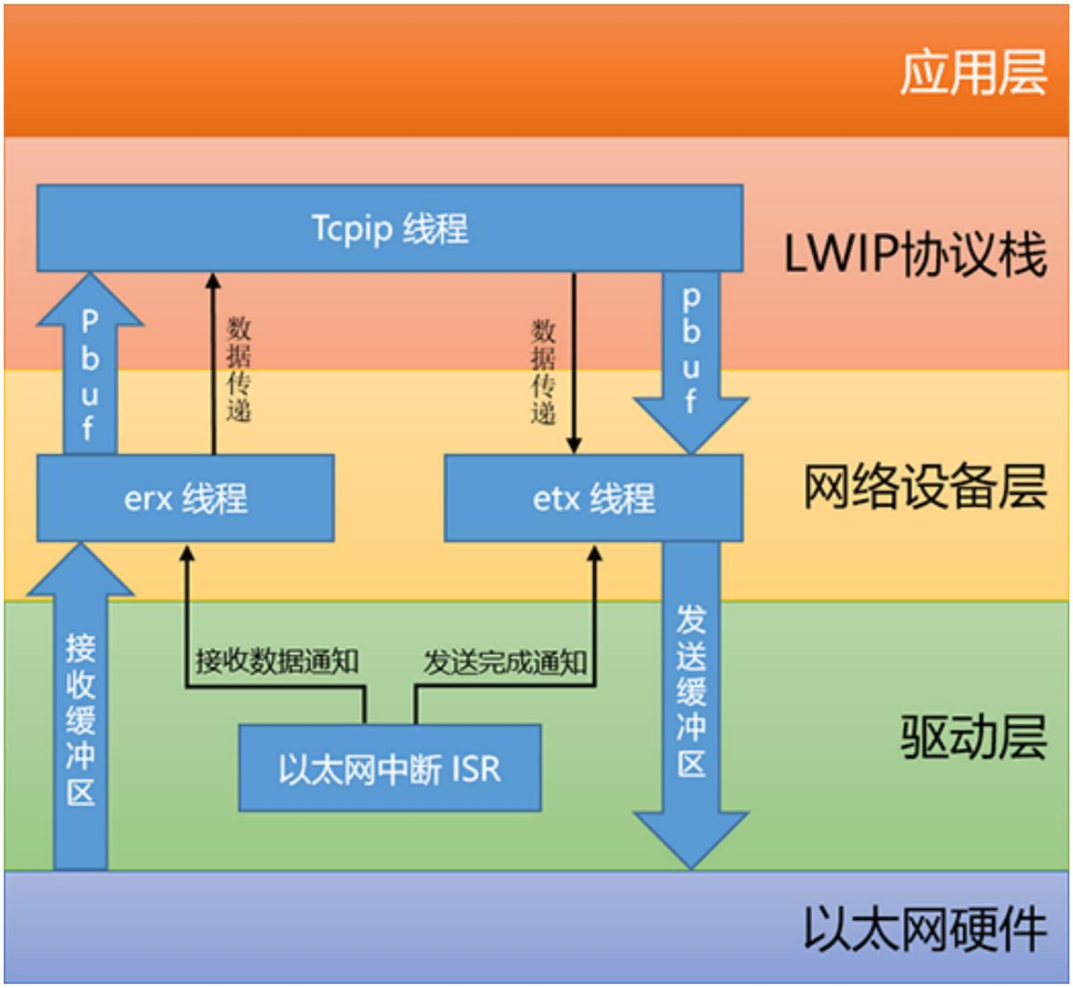

Figure **1:** *lwip_block*

RT-Thread's lwIP port builds on the original version, adding a network device layer to replace the original driver layer. Unlike the original driver layer, it uses an independent dual-threaded architecture for Ethernet data transmission and reception. Normally, the erx and etx threads have the same priority, but users can fine-tune this to prioritize transmission or reception based on their needs.

4.1 Data Receiving Process

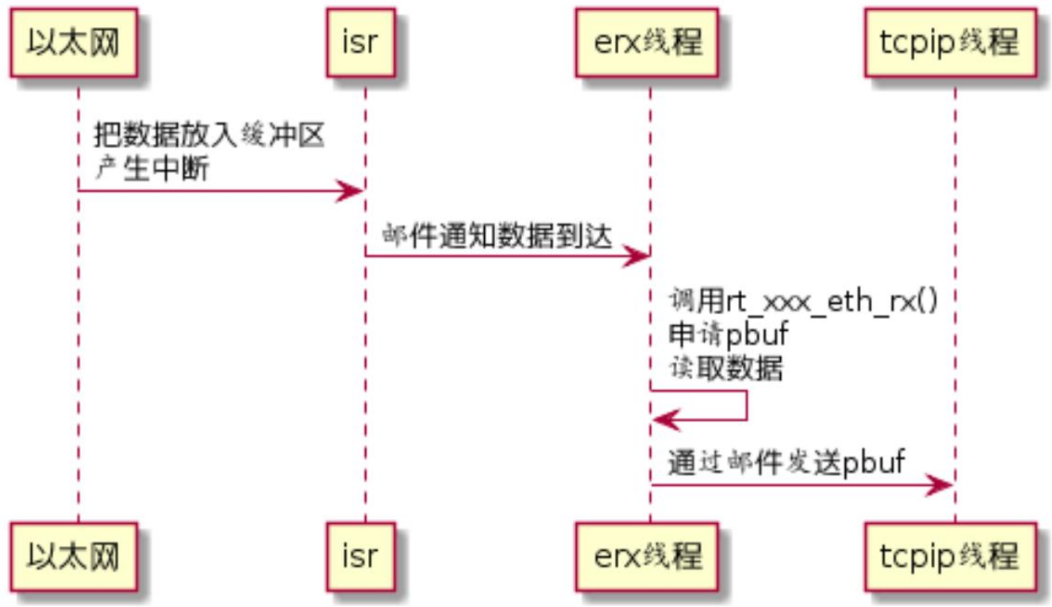

Figure 2: *rt_xxx_eth_rx*

When the Ethernet hardware device receives a network message and generates an interrupt, the received data will be stored in the receive buffer and then

The Ethernet interrupt program will then send an email to wake up the erx thread, which will then request a timeout based on the length of the received data.

pbuf, and put the data into the payload of the pbuf, and then send the pbuf to be processed by email.

Machine Translated by Google

4.2 Data sending process

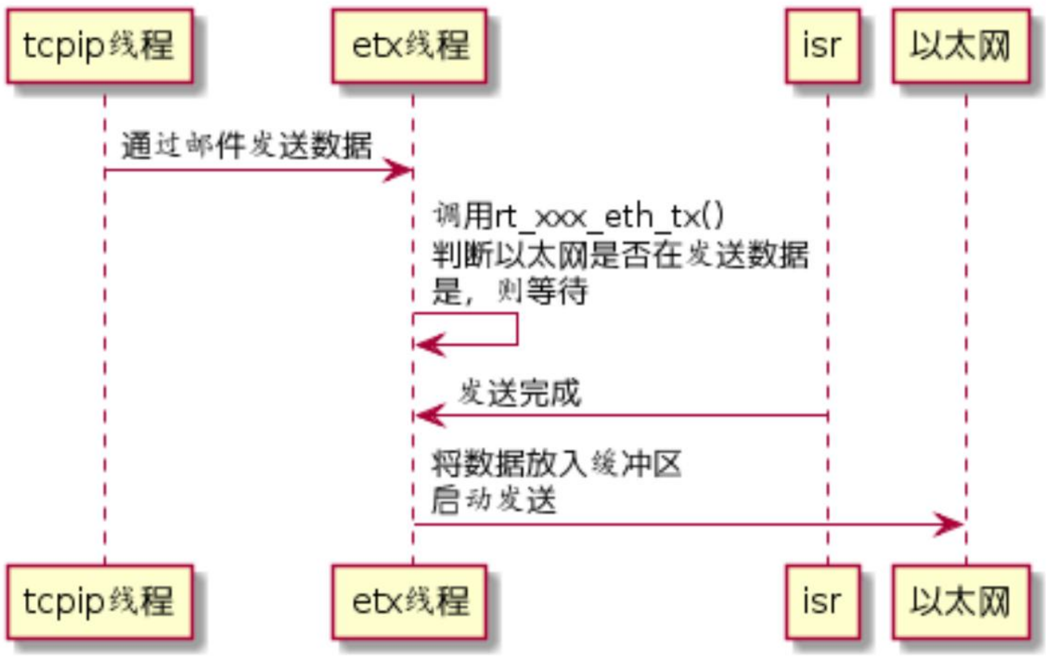

Figure **3:** *rt_xxx_erh_tx*

When data needs to be sent, the TCPIP thread sends an email to wake up the etx thread. The etx thread first determines whether data is currently being sent. If not, the etx thread places the data to be sent into the send buffer and then initiates Ethernet transmission. If data is currently being sent, the etx thread suspends itself until it receives the send wait semaphore.

Note: Under certain conditions, the etx/erx tasks added to RT-Thread can also be removed. When RX_THREAD is removed, other threads or interrupts are required to submit the received pbuf data packets to the lwIP main task. This is not described in detail here.

5. Introduction to Network Equipment

RT-Thread network devices inherit standard devices and are defined by the eth_device structure.

The eth_device structure code is provided for reference

```
struct eth_device {

    /* Standard equipment */
    struct rt_device parent;

    /* lwIP network interface */
```

```
    struct netif *netif; /* Send
    response signal*/
    struct rt_semaphore tx_ack;

    /* Network status flag*/
    rt_uint16_t flags; rt_uint8_t
    link_changed; rt_uint8_t link_status;


    /* Data packet receiving and
    sending interface*/ struct pbuf* (*eth_rx)(rt_device_t dev);
    rt_err_t (*eth_tx)(rt_device_t dev, struct pbuf* p);
};
```

This article uses the example of stm32f40x as an example to explain the relatively complete Ethernet driver. In addition to the ST firmware library, the following needs to be implemented drive:

5.1 Standard device interface

The standard device interface needs to be provided to the parent element in the eth_device structure

```
static rt_err_t rt_stm32_eth_init(rt_device_t dev); static rt_err_t
rt_stm32_eth_open(rt_device_t dev, rt_uint16_t oflag); static rt_err_t rt_stm32_eth_close(rt_device_t
dev); static rt_size_t rt_stm32_eth_read(rt_device_t dev, rt_off_t pos, void*

    buffer, rt_size_t size);
static rt_size_t rt_stm32_eth_write (rt_device_t dev, rt_off_t pos, const
     void* buffer, rt_size_t size);
static rt_err_t rt_stm32_eth_control(rt_device_t dev, int cmd, void *args
    );
```

rt_stm32_eth_init is used to initialize the DMA and MAC controllers.

rt_stm32_eth_open is used by upper-layer applications to open network devices. It is not currently used and directly returns RT_EOK.

rt_stm32_eth_close is used by upper-layer applications to close network devices. It is currently not used and directly returns RT_EOK.

rt_stm32_eth_read is used for upper-layer applications to directly read and write to the underlying device. For network devices, each message has a fixed format, so this interface is not currently used and directly returns a value of 0.

rt_stm32_eth_write is used for upper-layer applications to directly read and write to the underlying device. For network devices, each message has a fixed format, so this interface is not currently used and directly returns a value of 0.

rt_stm32_eth_control is used to control the Ethernet interface device. It is currently used to obtain the MAC address of the Ethernet interface. If necessary, other control functions can be expanded by adding control words.

5.2 Data packet sending and receiving interface

Corresponding to the eth_rx and eth_tx elements in the eth_device structure , it implements the data packet sending and receiving function, as shown below

Show:

```
rt_err_t rt_stm32_eth_tx( rt_device_t dev, struct pbuf* p); struct pbuf
*rt_stm32_eth_rx(rt_device_t dev);
```

The rt_stm32_eth_tx function is called by the etx thread to implement the data sending function. Here is the stm32f40x

Some code snippets are provided for reference

```
rt_err_t rt_stm32_eth_tx( rt_device_t dev, struct pbuf* p) {

    ......

    /* Determine if Ethernet is sending data*/
    while ((DMATxDescToSet->Status & ETH_DMATxDesc_OWN) != (uint32_t)
        RESET
    {
        rt_err_t result;
        rt_uint32_t level;

        /* Enter the sending waiting state*/
        level = rt_hw_interrupt_disable(); tx_is_waiting =
        RT_TRUE;
        rt_hw_interrupt_enable(level);

        /* Wait for the send completion semaphore, which will be released in the interruption*/
        result = rt_sem_take(&tx_wait, RT_WAITING_FOREVER); if (result ==
        RT_EOK) break; if (result ==
        -RT_ERROR) return -RT_ERROR;
    }

    /* Put the data in pbuf into the send buffer*/
    offset = 0; for
    (q = p; q != NULL; q = q->next) {

        uint8_t *to;

        to = (uint8_t*)((DMATxDescToSet->Buffer1Addr) + offset); memcpy(to, q->payload,
        q->len); offset += q->len;

    }
```

```
    /* Start sending */
    ......

}
```

The rt_stm32_eth_rx function is called by the erx thread to implement the function of receiving data. Here are some code snippets of stm32f40x for reference

```
struct pbuf *rt_stm32_eth_rx(rt_device_t dev) {

    ......

    /* Get frame length*/
    framelength = ((DMARxDescToGet->Status & ETH_DMARxDesc_FL) >>
        ETH_DMARXDESC_FRAME_LENGTHSHIFT) - 4;

    /* Apply for pbuf */
    p = pbuf_alloc(PBUF_LINK, framelength, PBUF_RAM); if (p !=
    RT_NULL) {

        struct pbuf* q;

        /* Copy the received data to pbuf*/
         for (q = p; q != RT_NULL; q= q->next) {

            memcpy(q->payload, (uint8_t *)((DMARxDescToGet->Buffer1Addr)
                + offset), q->len);
            offset += q->len;
         }
    }

    ......
    /* Return pbuf pointer */
    return p;
}
```

5.3 Driver initialization entry

```
void rt_hw_stm32_eth_init(void);
```

rt_hw_stm32_eth_init is used to register Ethernet devices, Ethernet hardware, configure MAC addresses, etc.

Machine Translated by Google

6. Transplantation Preparation

• Download RT-Thread source code

• Download the env tool

• Open env and enter the rt-thread/bsp/stm32f40x directory • Enter set RTT_CC=keil in the env command line to set the tool chain type to keil • Enter menuconfig in the env command line to enter the configuration interface and use the menuconfig tool to configure the project.

– Modify the console output to the serial port number of your own board – Enable lwIP

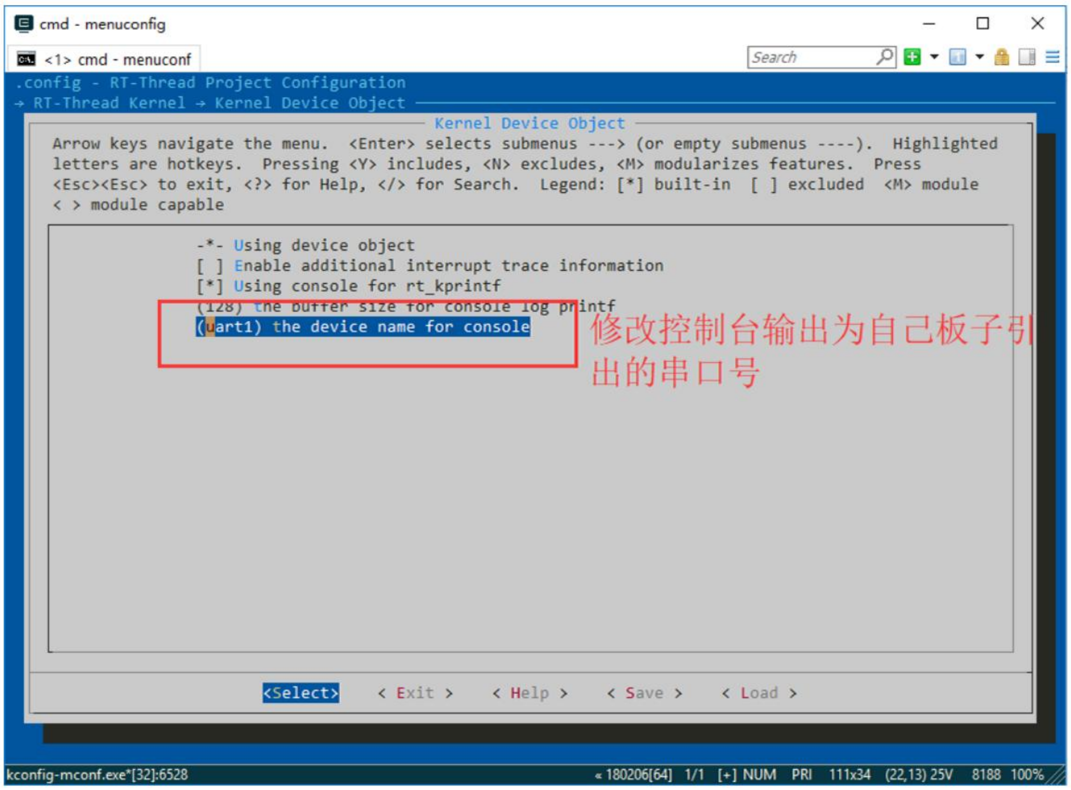

Figure 4: *ENV_uart*

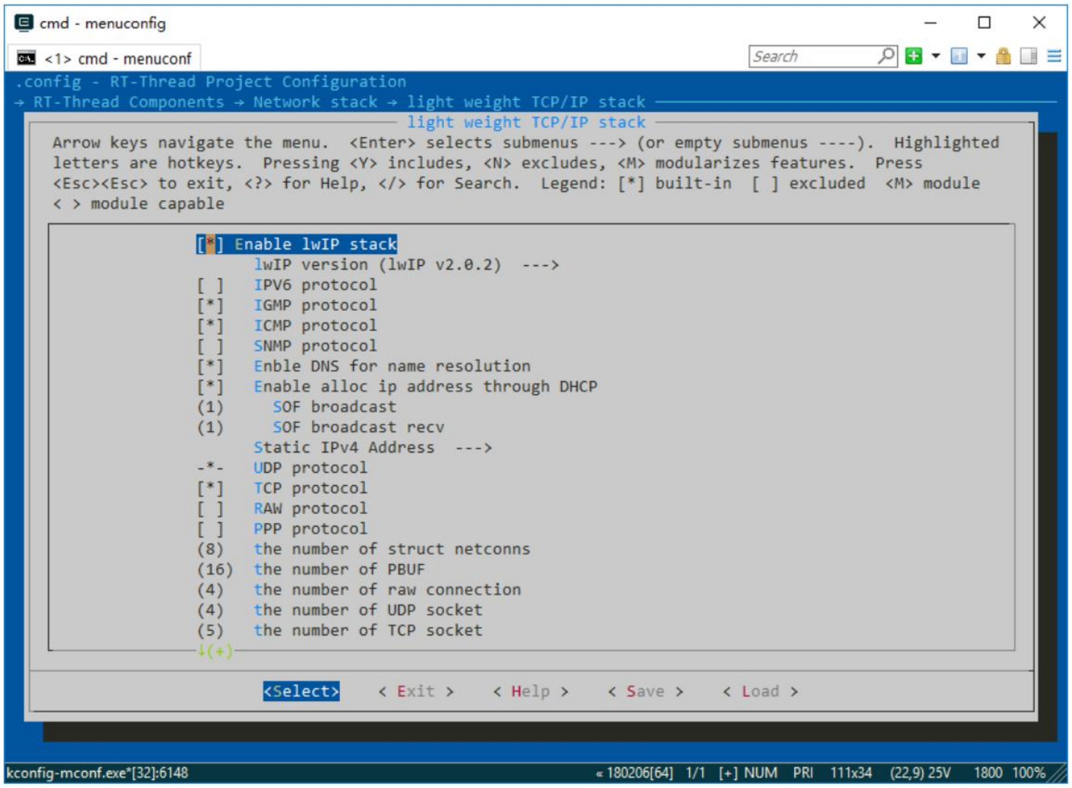

Figure **5:** *ENV_lwIP*

• Enter the command scons –target=mdk5 -s to generate the mdk5 project. • Open the project and open the stm32f4xx_eth.c file.

7 Driver Porting

The STM32F407 chip has its own Ethernet module, which includes a MAC 802.3 (Media Access Control) controller with a dedicated DMA controller, supports the Media Independent Interface (MII) and Reduced Media Independent Interface (RMII), and has an SMI interface for external PHY communication. Through a set of configuration registers, users can select the required mode and function for the MAC controller and DMA controller.

Because the MAC controller initialization of the same series of stm32 is basically the same, the driver MAC part of the same series The code does not need to be modified. Only the PHY part needs to be modified. Here we take LAN8720 as an example.

7.1 Data Packet Printing

When debugging the driver, it is recommended to enable the logging function in stm32f4xx_eth.c first.

```
/* debug settings*/ #define
ETH_DEBUG #define
ETH_RX_DUMP
```

```
#define ETH_TX_DUMP
```

7.2 Update PHY reset pin

Check the schematic diagram, the RESET pin is PD3, change the reset pin to PD3.

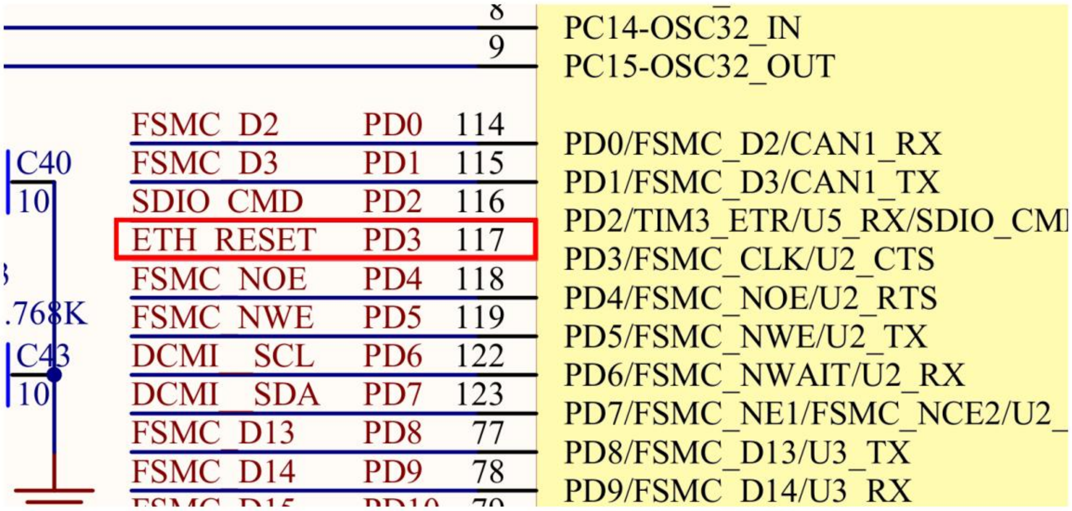

Figure **6:** *reset_pin*

```
void rt_hw_stm32_eth_init(void) {

    {
        GPIO_InitTypeDef GPIO_InitStructure;

        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
        GPIO_Init(GPIOD, &GPIO_InitStructure);

        GPIO_ResetBits(GPIOD, GPIO_Pin_3);
        rt_thread_delay(2);
        GPIO_SetBits(GPIOD, GPIO_Pin_3);
        rt_thread_delay(2);
    }
}
```

In the rt_hw_stm32_eth_init function, GPIO is used to reset the PHY chip.

In addition to the PHY not being reset correctly, the correct pinout may also cause pin conflicts and damage the board.

7.3 Confirm MII/RMII Mode

According to the schematic diagram, confirm whether the external PHY uses MII or RMII mode.

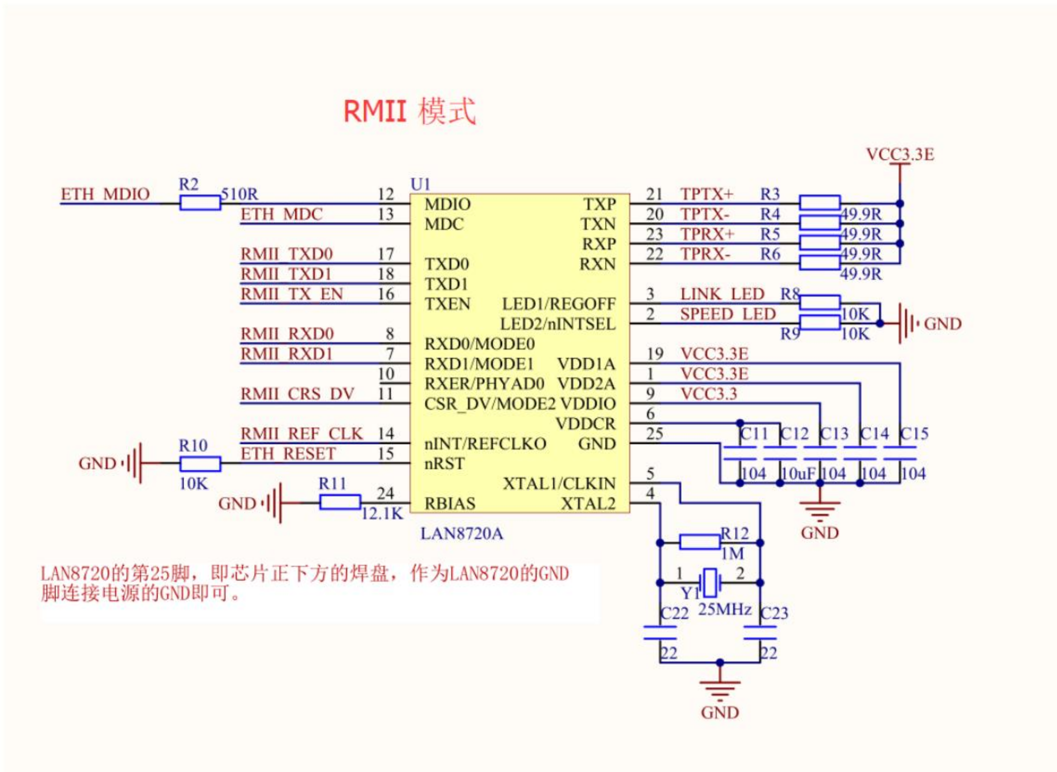

Figure **7:** *LAN8720*

Modify the mode settings in stm32f4xx_eth.c

```
#define RMII_MODE                    /* MII_MODE or RMII_MODE */
```

The driver uses the RMII mode which is simpler by default. When using the MII mode, it needs to be changed to

```
#define MII_MODE                    /* MII_MODE or RMII_MODE */
```

7.4 Confirm pin mapping

Many MCUs have pin mapping function for easy wiring. You need to determine the pins connected to the PHY according to the schematic diagram.

pin.

STM32F4 Ethernet, you can select different IO through IO mapping, the default is to select by group, punctual atomic

The Explorer uses GPIOG, so no code modification is required.

```
#define RMII_TX_GPIO_GROUP                2          /* 1:GPIOB or 2:GPIOG */
```

In fact, each IO of F4 can be set independently. If all TX IOs are not on the same GPIO port, such as TX1 on GPIOB and TX0 on GPIOG, then a large amount of code modification is required to ensure that each IO is mapped to the corresponding port.

In addition to data IO, the MDIO interface of some chips with many pins can also be mapped to different ports. This part of the configuration also needs to be checked one by one.

7.5 Pin Initialization

The following lists the pins used by the Atom Explorer to connect to the external PHY. Users need to initialize the corresponding pins according to their own schematics.

```
    /*
     ETH_MDIO ------------> PA2
     ETH_MDC -------------> PC1

     ETH_RMII_REF_CLK ----> PA1

     ETH_RMII_CRS_DV -----> PA7
     ETH_RMII_RXD0 -------> PC4
     ETH_RMII_RXD1 -------> PC5

     ETH_RMII_TX_EN ------> PG11
     ETH_RMII_TXD0 -------> PG13
     ETH_RMII_TXD1 -------> PG14

     ETH_RMII_TX_EN ------> PB11
     ETH_RMII_TXD0 -------> PB12
     ETH_RMII_TXD1 -------> PB13
     */

     GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_ETH); /*
            RMII_REF_CLK */
     GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_ETH); /* RMII_CRS_DV
              */

     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_7; GPIO_Init(GPIOA,
     &GPIO_InitStructure);
```

```
GPIO_PinAFConfig(GPIOC, GPIO_PinSource4, GPIO_AF_ETH); /* RMII_RXD0
    */
GPIO_PinAFConfig(GPIOC, GPIO_PinSource5, GPIO_AF_ETH); /* RMII_RXD1
    */

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
GPIO_Init(GPIOC, &GPIO_InitStructure);

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);

GPIO_PinAFConfig(GPIOG, GPIO_PinSource11, GPIO_AF_ETH); /* RMII_TX_EN
    */
GPIO_PinAFConfig(GPIOG, GPIO_PinSource13, GPIO_AF_ETH); /* RMII_TXD0
    */
GPIO_PinAFConfig(GPIOG, GPIO_PinSource14, GPIO_AF_ETH); /* RMII_TXD1
    */

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11 | GPIO_Pin_13 |
    GPIO_Pin_14;
GPIO_Init(GPIOG, &GPIO_InitStructure);
```

7.6 Update PHY Management Program

PHY is a standard module defined in IEEE802.3. The address space of PHY register is 5 bits, so the register

The register range is from 0 to 31, with a maximum of 32 registers. IEEE802.3 defines 16 basic registers with addresses 0-15.

The functions of the registers are retained, so only the definitions of a few registers need to be modified to complete the porting.

The driver uses LAN8720 by default. If you use other PHY models, you should modify the auto-negotiation, status and

The macro definitions related to the rate indication register enable the program to correctly set up auto-negotiation and read the connection status and rate of the PHY.

The following lists the common registers and control status bits of LAN8720 that need to be configured in stm32f4xx_eth.h

definition

```
#define PHY_BCR                     0                   /* Basic control registers */

/* Control bit definition of basic control register*/
#define PHY_Reset                   ((uint16_t)0x8000)          /* PHY soft
    reset*/
#define PHY_Loopback                ((uint16_t)0x4000)
#define PHY_FULLDUPLEX_100M         ((uint16_t)0x2100)
#define PHY_HALFDUPLEX_100M         ((uint16_t)0x2000)
#define PHY_FULLDUPLEX_10M          ((uint16_t)0x0100)
#define PHY_HALFDUPLEX_10M          ((uint16_t)0x0000)
```

```
#define PHY_AutoNegotiation            ((uint16_t)0x1000)      /* Enable
        Active negotiation*/
#define PHY_Restart_AutoNegotiation    ((uint16_t)0x0200)      /* Restart
        Automatic auto-negotiation*/
#define PHY_Powerdown                  ((uint16_t)0x0800)
#define PHY_Isolate                    ((uint16_t)0x0400)

#define PHY_BSR                         1          /* Basic status register */

/* Status bit definition of basic status register*/
#define PHY_AutoNego_Complete          ((uint16_t)0x0020)      /* Automatic coordination
        Business completed*/
#define PHY_Linked_Status */           ((uint16_t)0x0004)      /* Connection status

#define PHY_Jabber_detection detection ((uint16_t)0x0002)      /* jabber
        indication bit*/
```

In the RT-Thread driver implementation, a PHY address search function is implemented, which can correctly search for the PHY address. So there is no need to define the PHY address

Here is the address search function in the phy_monitor_thread_entry function in the stm32f4xx_eth.c file For user reference

```
/* Determine the address by whether the PHY chip ID number can be read*/
{
    rt_uint32_t i;
    rt_uint16_t temp;

    for(i=0; i<=0x1F; i++)
    {
        temp = ETH_ReadPHYRegister(i, 0x02);

        if( temp != 0xFFFF )
        {
            phy_addr = i;
            break;
        }
    }
}

/* PHY address not found*/
if(phy_addr == 0xFF)
{
    STM32_ETH_PRINTF("phy not probe!\r\n");
    return;
```

```
}
```

If the message "phy not probe!" is displayed, you should check the IO configuration and hardware.

Tip: Current mainstream PHYs typically default to auto-negotiation mode after reset, and current switches and network cables generally support 100M full-duplex. Therefore, before properly adapting the PHY, you can temporarily change the PHY operating mode to 100M full-duplex for testing purposes (by modifying the corresponding control bit in the basic control register).

7.7 Interrupt Function

Receive: When the interrupt function receives data, it will send an email to notify the "erx" thread to read the data.

Send: After sending is completed, the interrupt function will release the semaphore to tell the sending program that it can continue sending.

Here we only post the codes related to sending and receiving

```
status = ETH->DMASR;
ier = ETH->DMAIER;

if(status & ETH_DMA_IT_NIS) {

    rt_uint32_t nis_clear = ETH_DMA_IT_NIS;

    /* Send interrupt */
    if((status & ier) & ETH_DMA_IT_T) {

        STM32_ETH_PRINTF("ETH_DMA_IT_T\r\n");

        /* Sending is complete, release the send wait semaphore*/
        if (tx_is_waiting == RT_TRUE) {

            tx_is_waiting = RT_FALSE;
            rt_sem_release(&tx_wait);
        }

        nis_clear |= ETH_DMA_IT_T;
    }

    /* Receive interrupt */
    if((status & ier) & ETH_DMA_IT_R) /* packet reception */ {

        STM32_ETH_PRINTF("ETH_DMA_IT_R\r\n"); /* Send
        email to notify erx thread to receive data*/
        eth_device_ready(&(stm32_eth_device.parent));
```

```
            nis_clear |= ETH_DMA_IT_R;
        }

        /* Clear interrupt flag */
        ETH_DMAClearITPendingBit(nis_clear);
    }
```

7.8 ETH Device Initialization

RT-Thread real-time operating system provides a device management framework. Applications can use RT-Thread devices to manage The operation interface realizes the general device driver. Here we implement the Network Interface type device for ETH device. Then register it with RT-Thread.

rt_hw_stm32_eth_init in stm32f4xx_eth.c is the ETH device initialization entry, responsible for Initialize the stm32_eth_device structure and register it with RT-Thread.

```
/* Set working speed and mode*/
stm32_eth_device.ETH_Speed = ETH_Speed_100M;
stm32_eth_device.ETH_Mode = ETH_Mode_FullDuplex;

/* Set the MAC address using the STM32 global unique ID*/
stm32_eth_device.dev_addr[0] = 0x00;
stm32_eth_device.dev_addr[1] = 0x80;
stm32_eth_device.dev_addr[2] = 0xE1;
stm32_eth_device.dev_addr[3] = *(rt_uint8_t*)(0x1FFF7A10+4);
stm32_eth_device.dev_addr[4] = *(rt_uint8_t*)(0x1FFF7A10+2);
stm32_eth_device.dev_addr[5] = *(rt_uint8_t*)(0x1FFF7A10+0);

/* Set standard driver interface */
stm32_eth_device.parent.parent.init            = rt_stm32_eth_init;
stm32_eth_device.parent.parent.open            = rt_stm32_eth_open;
stm32_eth_device.parent.parent.close           = rt_stm32_eth_close;
stm32_eth_device.parent.parent.read            = rt_stm32_eth_read;
stm32_eth_device.parent.parent.write           = rt_stm32_eth_write;
stm32_eth_device.parent.parent.control         = rt_stm32_eth_control;
stm32_eth_device.parent.parent.user_data = RT_NULL;

/* Set the network driver receiving and sending interface*/
stm32_eth_device.parent.eth_rx        = rt_stm32_eth_rx;
stm32_eth_device.parent.eth_tx        = rt_stm32_eth_tx;

/* Initialize the sending signal */
rt_sem_init(&tx_wait, "tx_wait", 0, RT_IPC_FLAG_FIFO);
```

```
/* Register network
device */ eth_device_init(&(stm32_eth_device.parent), "e0");
```

7.8.1. Setting the standard driver interface

7.8.1.1. rt_stm32_eth_init rt_stm32_eth_init is used to initialize the Ethernet peripherals and should be initialized according to actual needs.

Here we only post the code for MAC configuration and DMA configuration

```
    /*------------------------MAC
        -----------------------------------*/
    ETH_InitStructure.ETH_AutoNegotiation = ETH_AutoNegotiation_Enable;
    ETH_InitStructure.ETH_Speed = stm32_eth->ETH_Speed;
    ETH_InitStructure.ETH_Mode = stm32_eth->ETH_Mode;
    ETH_InitStructure.ETH_LoopbackMode = ETH_LoopbackMode_Disable;
    ETH_InitStructure.ETH_RetryTransmission =
        ETH_RetryTransmission_Disable;
    ETH_InitStructure.ETH_AutomaticPadCRCStrip =
        ETH_AutomaticPadCRCStrip_Disable;
    ETH_InitStructure.ETH_ReceiveAll = ETH_ReceiveAll_Disable;
    ETH_InitStructure.ETH_BroadcastFramesReception =
        ETH_BroadcastFramesReception_Enable;
    ETH_InitStructure.ETH_PromiscuousMode = ETH_PromiscuousMode_Disable;
    ETH_InitStructure.ETH_MulticastFramesFilter =
        ETH_MulticastFramesFilter_HashTable;
    ETH_InitStructure.ETH_HashTableHigh = stm32_eth->ETH_HashTableHigh;
    ETH_InitStructure.ETH_HashTableLow = stm32_eth->ETH_HashTableLow;
    ETH_InitStructure.ETH_UnicastFramesFilter =
        ETH_UnicastFramesFilter_Perfect;
#ifdef CHECKSUM_BY_HARDWARE
    ETH_InitStructure.ETH_ChecksumOffload = ETH_ChecksumOffload_Enable;
#endif

    /*------------------------DMA
        -----------------------------------*/
    ETH_InitStructure.ETH_DropTCPIPChecksumErrorFrame =
        ETH_DropTCPIPChecksumErrorFrame_Enable;
    ETH_InitStructure.ETH_ReceiveStoreForward =
        ETH_ReceiveStoreForward_Enable;
    ETH_InitStructure.ETH_TransmitStoreForward =
        ETH_TransmitStoreForward_Enable;
```

```
    ETH_InitStructure.ETH_ForwardErrorFrames =
        ETH_ForwardErrorFrames_Disable;
    ETH_InitStructure.ETH_ForwardUndersizedGoodFrames =
        ETH_ForwardUndersizedGoodFrames_Disable;
    ETH_InitStructure.ETH_SecondFrameOperate =
        ETH_SecondFrameOperate_Enable;
    ETH_InitStructure.ETH_AddressAlignedBeats =
        ETH_AddressAlignedBeats_Enable;
    ETH_InitStructure.ETH_FixedBurst = ETH_FixedBurst_Enable;
    ETH_InitStructure.ETH_RxDMABurstLength = ETH_RxDMABurstLength_32Beat;
    ETH_InitStructure.ETH_TxDMABurstLength = ETH_TxDMABurstLength_32Beat;
    ETH_InitStructure.ETH_DMAArbitration =
        ETH_DMAArbitration_RoundRobin_RxTx_2_1;

    /* Configure Ethernet peripherals */
    ETH_Init(&ETH_InitStructure);

    /* DMA interrupt setting */
    ETH_DMAITConfig(ETH_DMA_IT_NIS | ETH_DMA_IT_R | ETH_DMA_IT_T, ENABLE)
        ;

    /* Initialize the descriptor list */
    ETH_DMATxDescChainInit(DMATxDscrTab, &Tx_Buff[0][0], ETH_TXBUFNB);
    ETH_DMARxDescChainInit(DMARxDscrTab, &Rx_Buff[0][0], ETH_RXBUFNB);

    /* DMA address setting*/
    ETH_MACAddressConfig(ETH_MAC_Address0, (u8*)&stm32_eth_device.
        dev_addr[0]);
```

7.8.1.2. rt_stm32_eth_control The rt_stm32_eth_control function needs to implement the function of obtaining the MAC address

```
static rt_err_t rt_stm32_eth_control(rt_device_t dev, int cmd, void *args
    )
{
    switch(cmd) {

    case NIOCTL_GADDR: /
        * Get MAC address */
        if(args) rt_memcpy(args, stm32_eth_device.dev_addr, 6); else return
        -RT_ERROR; break;


    default :
```

```
            break;
        }

        return RT_EOK;
}
```

Other settings of the standard driver interface do not need to be implemented, just write an empty function first.

7.8.2. Data packet sending and receiving interface

7.8.2.1. rt_stm32_eth_rx

rt_stm32_eth_rx will read the data in the receive buffer, put it into pbuf (lwIP uses the structure pbuf to manage data packets), and return the pbuf pointer.

The "erx" receiving thread will block on getting the eth_rx_thread_mb mailbox, and when it receives the mail, it will call Use rt_stm32_eth_rx to receive data.

7.8.2.2. rt_stm32_eth_tx

rt_stm32_eth_tx puts the data to be sent into the transmit buffer and waits for DMA to send the data.

The "etx" sending thread will be blocked on getting the eth_tx_thread_mb mailbox. When it receives the email, it will call Use rt_stm32_eth_tx to send data.

8 EMAC driver debugging

8.1 Experimental Environment Construction

The DHCP function is enabled by default in the project, and a DHCP server is required to assign IP addresses. Common connection expansion

As shown in the picture:

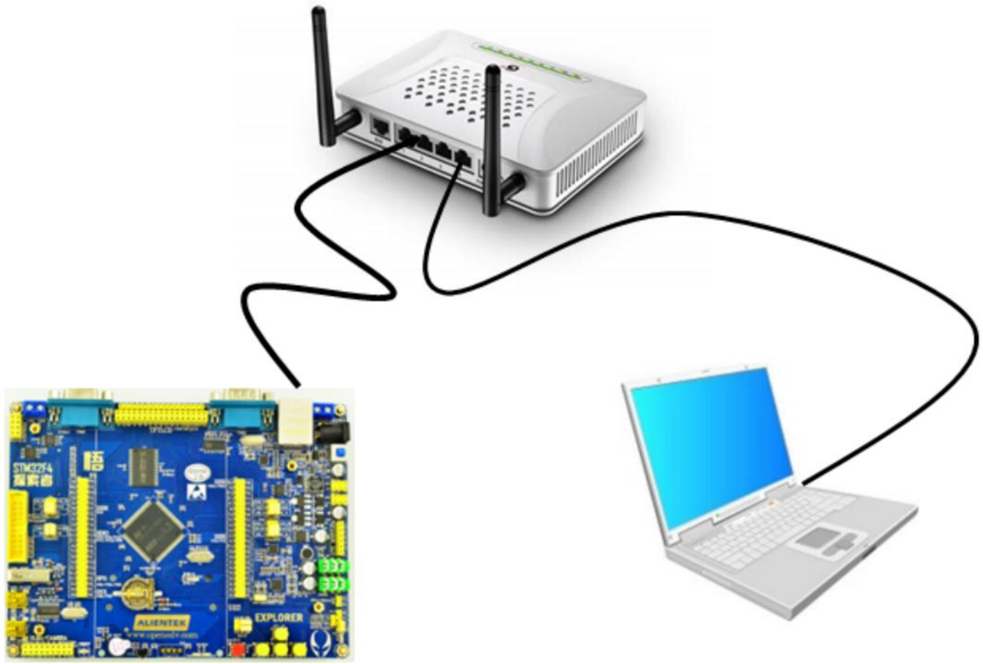

Figure **8:** *eth_RJ45*

If there is no convenient actual environment, you can also configure a fixed IP through ENV first, and then connect directly to the debugger with a network cable.

Trial computer.

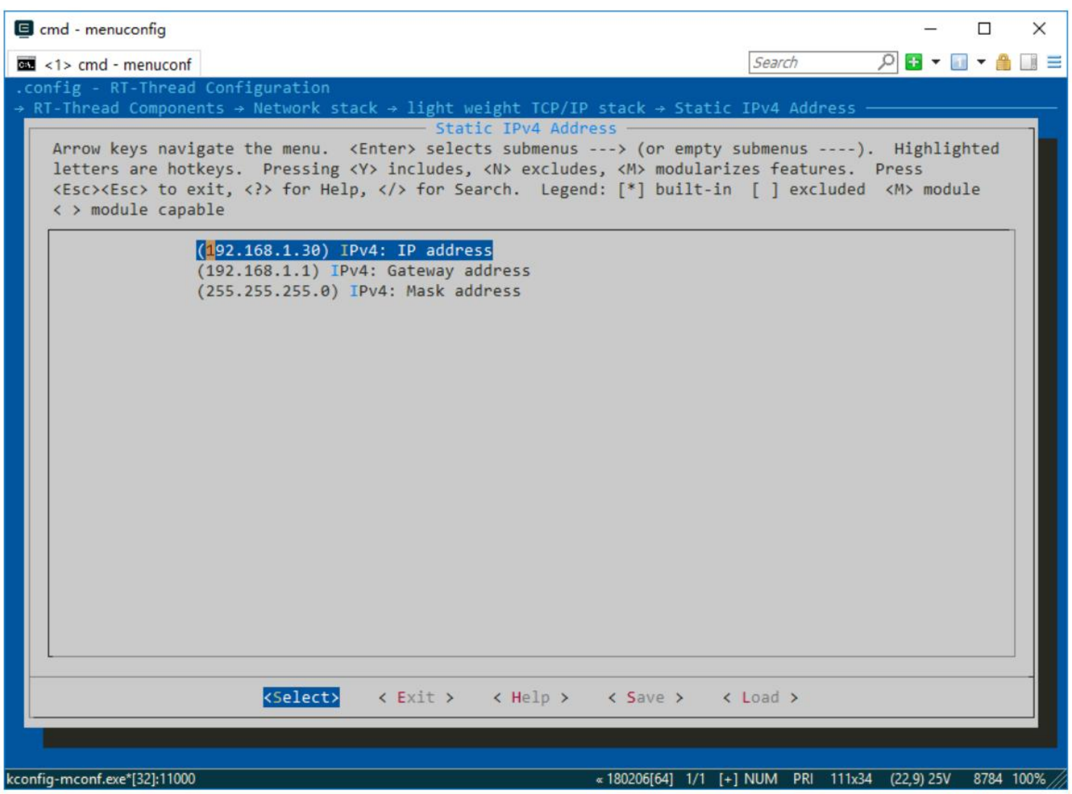

Figure **9:** *iPadress*

The computer and development board need to be set to the same IP address segment.

8.2 Confirm PHY connection status

When the network-enabled firmware is running on the development board, the first thing you should do is check the status of the RJ45 indicator light.

Normally the light should be on and it will flash when data is being sent or received.

If the light is off, please first confirm whether the development board hardware is intact; then check whether the power supply is sufficient and whether the network cable is connected properly.

Then, the program cooperates with the hardware to confirm whether the PHY is reset correctly until it flashes normally.

8.3 Confirm IP address

Execute the ifconfig command in the shell command line to view the IP address of the network card.

If the DHCP function is enabled and it is displayed that an IP address has been obtained, it means that the driver's sending and receiving functions have been

Normal menstruation.

If the IP address is static or no IP address has been obtained, please pay attention to the UP and LINK_UP flags in the network card flags. If LINK_DOWN is displayed, please confirm that the PHY management program correctly identifies the PHY speed and duplex state and correctly notifies lwIP through eth_device_linkchange.

If the DHCP function is enabled and LINK_UP is displayed, but the IP address cannot be obtained correctly, the development board

Communication with the DHCP server is not working properly and further debugging is required.

```
msh />ifconfig
network interface: e0 (Default)
MTU: 1500
MAC: 00 04 9f 05 44 e5
FLAGS: UP LINK_UP ETHARP
ip address: 192.168.12.127
gw address: 192.168.10.1
net mask  : 255.255.0.0
dns server #0: 192.168.10.1
dns server #1: 223.5.5.5
msh />
```

Figure **10:** *ifconfig*

You can check whether the PHY register value is read correctly by plugging and unplugging the network cable and observing the LINK status.

8.4 Printing Data Packets

By opening the ETH_RX_DUMP and ETH_TX_DUMP functions in the driver, you can

The package is printed out.

```
tx_dump, len:350
ff ff ff ff ff ff 00 80   e1 0c 26 27 08 00 45 00
01 50 00 05 00 00 ff 11   ba 98 00 00 00 00 ff ff
ff ff 00 44 00 43 01 3c   1d ca 01 01 06 00 56 47
a0 b0 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 80   e1 0c 26 27 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 63 82   53 63 35 01 01 39 02 05
dc 37 04 01 03 1c 06 ff   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00
dump done!
```

Figure **11:** *dump*

When the RJ45 indicator light flashes normally, it means that data packets are being sent and received. Because there are often broadcast data packets in the network, If there are data packets coming in at this time, there will be RX_DUMP printed. If there is no printing, please check the following two points:

- There is a problem with the RX line of the MII/RMII, including hardware problems or IO mapping errors.
- The rate and duplex mode of EMAC and PHY are not configured, for example, EMAC works at 10M and PHY is connected at 100M.

You need to confirm that you have correctly obtained the PHY rate and duplex mode, and compare it with the other end of the network cable. For example, if the computer shows 10M, but the PHY management program has not been updated on the board, the default is 100M.

If necessary, you can also print the PHY's LOOPBACK function to confirm that the MII/RMII bus is intact. of.

8.5 Ping Test

You can ping the board from your computer or the board can ping your computer (you need to enable the netutils component package). ping function) to test whether the driver is successfully transplanted.

```
C:\Users\yhd>ping 192.168.12.127

正在 Ping 192.168.12.127 具有 32 字节的数据:
来自 192.168.12.127 的回复: 字节=32 时间<1ms TTL=255
来自 192.168.12.127 的回复: 字节=32 时间<1ms TTL=255
来自 192.168.12.127 的回复: 字节=32 时间<1ms TTL=255
来自 192.168.12.127 的回复: 字节=32 时间<1ms TTL=255

192.168.12.127 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms
```

Figure **12:** *ping1*

```
msh />ping 192.168.12.45
60 bytes from 192.168.12.45 icmp_seq=0 ttl=128 time=1 ticks
60 bytes from 192.168.12.45 icmp_seq=1 ttl=128 time=0 ticks
60 bytes from 192.168.12.45 icmp_seq=2 ttl=128 time=0 ticks
60 bytes from 192.168.12.45 icmp_seq=3 ttl=128 time=0 ticks
```

Figure **13:** *ping2*

Before pinging, please note the following:

• Some PHY chips require additional initialization according to the manual. • The board

and computer must be in the same network segment.

• If the computer is connected to both the network cable and Wi-Fi, the board will not be pingable. • Check if the

firewall has disabled the ping function. • Some corporate

networks prohibit the ping function; it is recommended to change the network environment. •

The MAC address is not standardized, and the board cannot ping the external network.

• If the erx stack size is set too small, the erx thread stack will overflow.

8.6 **Wireshark** Packet Capture

If the board has RX DUMP but still cannot communicate, you can use wireshark on the computer Capture packets.

The computer pings the development board, and the development board receives a request packet with its own IP address as the target address. The child will respond to the computer by sending a reply packet with the destination address being the computer's IP address.

Depending on whether there is a reply, there are two situations to check.

• Whether the board has received the request packet (request).

• Whether the board has sent a response (reply).

If there is a response but the ping is unsuccessful, you can check whether the packet content meets the specifications.

8.6.1. Filter by **MAC** address

When the board sends a broadcast packet, it can be received on the computer. (such as DHCP Discoverer)

When using wireshark to capture packets, we mainly use various filters flexibly to filter out the data we are concerned about. Bag.

When the development board has not yet obtained an IP address, you can use the MAC address of the development board as a filter condition.

Machine Translated by Google

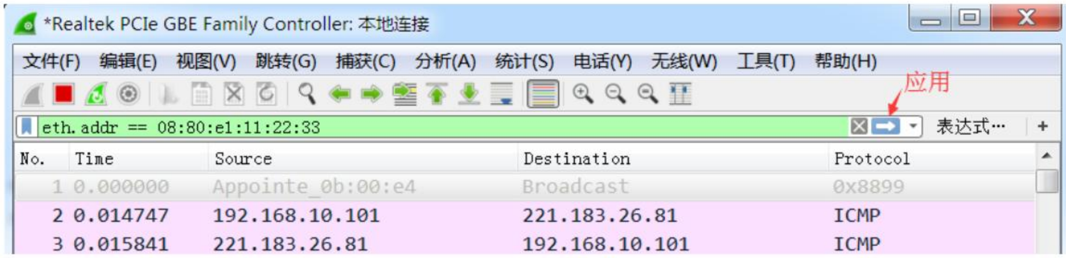

Figure **14:** *wireshark_mac*

When the development board successfully links up, it will actively send out a DHCP request packet, and the source address is the development board's own MAC address.

8.6.2. Filtering by **IP** Address

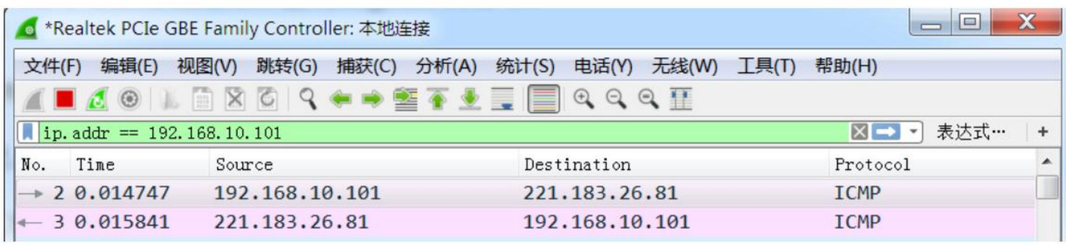

Figure **15:** *wireshark_ip*

Or configure the development board to a static IP address, and then execute the ping command on the PC to ping the IP address of the development board.

Here, the IP address of the development board is used as the filtering condition. Under normal circumstances, the PC will send a request packet first.