试验原因

我移植了 STM32F4 + ucosii + lwip + lan8720, 编译过了，发现网卡ping不通。
单步发现，网卡初始化都没过. 卡死在下面的实现

```
1    while (ETH_GetSoftwareResetStatus() == SET);
```

通过单步能正常运行的第三方工程，发现我的试验工程GPIO初始化错了。en.stsw-stm32070的PHY是DP83848CVV， 用的是MII接口。我试验的板子是LAN8720, 用的是RMII接口。软件上要采用RMII接口，且GPIO初始化时，只能初始化RMII指定的GPIO, MII相关的接口不能去初始化。

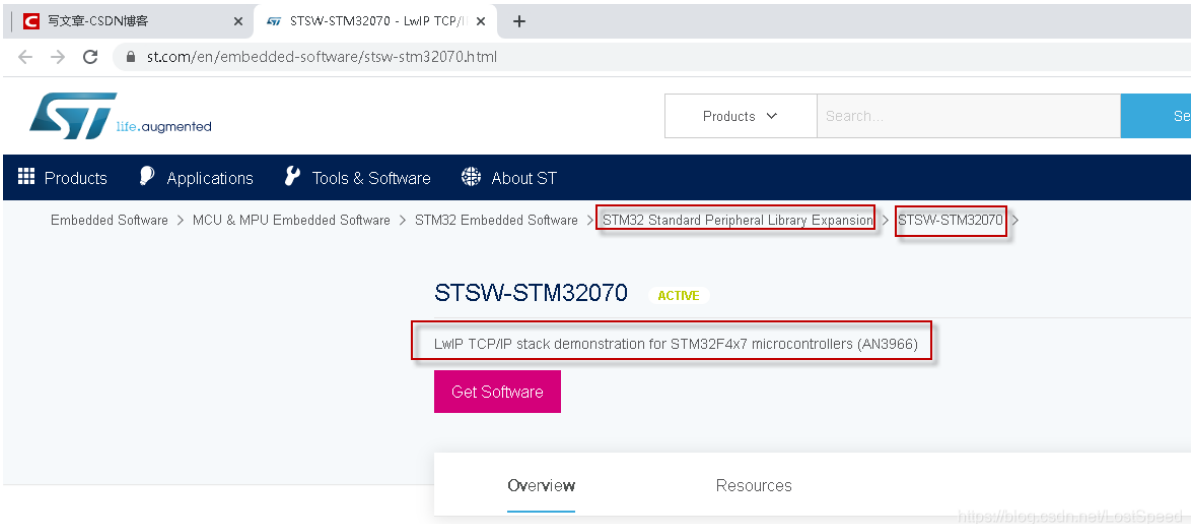
看了下游开发板厂商的资料，虽然能运行，但是没讲为啥那么改，具体改的是哪里，要改那几个点?

这就不知道具体的修改点了，没多大意义。
如果不知道人家改了哪，那样就迁移过来用，好乱的。

这要是用起来，以后出点问题，那咋弄...

看了下游厂商的工程，都是从ST官方的demo改出来的。
只是有的店家改的简洁(只作必要的修改)和官方demo很像。
有的店家加了自己的组件，多此一举，对开发者帮助不大。
demo就是要突出知识点，加那么多杂七杂八的内容，将知识点盖住了。

官方F4固件库lwip的demo是 en.stsw-stm32070



去下载了ST官方F4的lwip工程，结合能在板子上跑的第三方工程，想弄清楚，如何能从官方demo上，改出一个能在板子上跑的程序。

改好的工程

STM32F4x7_ETH_LwIP_V1.1.1_modify_phy_to_lan8720_2020_0307_2039.zip

试验

STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver 是无操作系统的demo工程, 将这个demo搞懂，网卡的基本响应(e.g. ICMP)就没问题, 剩下的事就是加rtos, 配置lwip的参数, 然后起个任务收包，回包就O了。

我手头有STM3240G-EVAL, 能跑这个demo.

但是产品板子上是F407 + LAN8720, 用的RMII接口，硬件电路连接是以前同事从第三方厂商的原理图中摘出来的。我必须将这个demo工程改成LAN8720的PHY, 在 F407 + LAN8720的板子上跑起来，才能解决产品板子上以后可能遇到的问题。

en.stsw-stm32070工程的修改点

修改接口为RMII接口

en.stsw-stm32070用的DP83848CVV是MII接口，用的硬件连线比较多.
开发板用的PHY是LAN8720, 是RMII接口，用的硬件连线比较少。
为了使用LAN8720, PHY接口方式必须改为RMII

\STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver\inc\main.h

```
/*Static IP ADDRESS: IP_ADDR0.IP_ADDR1.IP_ADDR2.IP_ADDR3 */
// 本机静态IP, 根据试验环境修改
#define IP_ADDR0   192
#define IP_ADDR1   168
#define IP_ADDR2   1
#define IP_ADDR3   10

/*NETMASK*/
#define NETMASK_ADDR0   255
#define NETMASK_ADDR1   255
#define NETMASK_ADDR2   255
#define NETMASK_ADDR3   0

/*Gateway Address*/
// 局域网的网关, 根据试验环境修改
#define GW_ADDR0   192
#define GW_ADDR1   168
#define GW_ADDR2   1
#define GW_ADDR3   1

/* MII and RMII mode selection, for STM324xG-EVAL Board(MB786) RevB ***********/
// 定义RMII_MODE宏
#define RMII_MODE  // User have to provide the 50 MHz clock by soldering a 50 MHz
                     // oscillator (ref SM7745HEV-50.0M or equivalent) on the U3
                     // footprint located under CN3 and also removing jumper on
JP5.
                     // This oscillator is not provided with the board.
                     // For more details, please refer to STM3240G-EVAL
evaluation
                     // board User manual (UM1461).

// 注释掉MII_MODE宏
```

```
// #define MII_MODE

/* Uncomment the define below to clock the PHY from external 25MHz crystal (only for MII mode) */
#ifdef  MII_MODE
 #define PHY_CLOCK_MCO
#endif


```

修改PHY地址

因为硬件连接不同，我试验板子上PHYADDR0是悬空的，默认地址是0
有些不存在的MII函数，也注释掉
\STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver\inc\stm32f4x7_eth_bsp.h

```
#include "netif.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
// RMII可以支持多(32)个PHY
// 修改为和实际PHY地址一样的值，才能让RMII选中LAN8720作为当前PHY
#define LAN8720_PHY_ADDRESS       ((uint16_t) 0x00) // PHYAD0引脚在硬件上是悬空的，默认就是PHY地址默认是0

/* Specific defines for EXTI line, used to manage Ethernet link status */
#define ETH_LINK_EXTI_LINE             EXTI_Line14
#define ETH_LINK_EXTI_PORT_SOURCE      EXTI_PortSourceGPIOB
#define ETH_LINK_EXTI_PIN_SOURCE       EXTI_PinSource14
#define ETH_LINK_EXTI_IRQn             EXTI15_10_IRQn
/* PB14 */
#define ETH_LINK_PIN                   GPIO_Pin_14
#define ETH_LINK_GPIO_PORT             GPIOB
#define ETH_LINK_GPIO_CLK              RCC_AHB1Periph_GPIOB

/* Ethernet Flags for EthStatus variable */
#define ETH_INIT_FLAG           0x01 /* Ethernet Init Flag */
#define ETH_LINK_FLAG           0x10 /* Ethernet Link Flag */

/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void  ETH_BSP_Config(void);
// 下面2个函数是MII接口(和RMII的PHY硬件连接也有关系)才需要的函数，不需要了
// uint32_t Eth_Link_PHYITConfig(uint16_t PHYAddress);
// void Eth_Link_EXTIConfig(void);
void Eth_Link_ITHandler(uint16_t PHYAddress);
void ETH_link_callback(struct netif *netif);

```

修改PHY寄存器参数

不同的PHY符合RMII接口的寄存器地址是不一样的

\STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver\inc\stm32f4x7_eth_conf.h

```
/******************  PHY Extended Registers section : ***********************/

/* These values are relatives to DP83848 PHY and change from PHY to another,
   so the user have to update this value depending on the used external PHY */

// LAN8720的3个寄存器参数
/* The PHY status register  */
#define PHY_SR                ((uint16_t)0x1F) /* PHY status register Offset */
#define PHY_SPEED_STATUS      ((uint16_t)0x0004) /* PHY Speed mask */
#define PHY_DUPLEX_STATUS     ((uint16_t)0x0010) /* PHY Duplex mask */

// 下面的寄存器参数是MII接口的DP83848才有的，注释掉
///* The DP83848 PHY: MII Interrupt Control Register  */
//#define PHY_MICR              ((uint16_t)0x11) /* MII Interrupt Control Register */
//#define PHY_MICR_INT_EN       ((uint16_t)0x0002) /* PHY Enable interrupts */
//#define PHY_MICR_INT_OE       ((uint16_t)0x0001) /* PHY Enable output interrupt events */

///* The DP83848 PHY: MII Interrupt Status and Misc. Control Register */
//#define PHY_MISR              ((uint16_t)0x12) /* MII Interrupt Status and Misc. Control Register */
//#define PHY_MISR_LINK_INT_EN  ((uint16_t)0x0020) /* Enable Interrupt on change of link status */
//#define PHY_LINK_STATUS       ((uint16_t)0x2000) /* PHY link status interrupt mask */

   /* Note : Common PHY registers are defined in stm32f4x7_eth.h file */

/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

```

将不用的设备初始化代码注释掉

这个试验只验证F407 + LAN8720是否能在en.stsw-stm32070工程上跑起来，将不用的设备初始化都去掉.

\STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver\src\main.c

```
int main(void)
{
  /*!< At this stage the microcontroller clock setting is already configured to
       168 MHz, this is done through SystemInit() function which is called from
       startup file (startup_stm32f4xx.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32f4xx.c file
     */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

#ifdef SERIAL_DEBUG
  DebugComPort_Init();
#endif
```

```
14
15    // 将LCD/LED等多余的设备初始化去掉.
16    /*Initialize LCD and Leds */
17    // LCD_LED_Init();
18
19    /* configure ethernet (GPIOs, clocks, MAC, DMA) */
20    ETH_BSP_Config();
21
22    /* Initilaize the LwIP stack */
23    LwIP_Init();
24
25    /* Http webserver Init */
26    httpd_init();
27
```

修改PHY驱动

不同PHY的寄存器参数读取方式不同
注释掉LAN8720没有的寄存器操作
\STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver\src\stm32f4x7_eth_bsp.c

```
1   void ETH_BSP_Config(void)
2   {
3     // 新建2个变量，保存从PHY寄存器读出的值，作判断网卡上线的判断.
4     uint16_t rc1 = 0;
5     uint16_t rc2 = 0;
6
7     RCC_ClocksTypeDef RCC_Clocks;
8
9       /*************************************************************************
10      NOTE:
11           When using Systick to manage the delay in Ethernet driver, the Systick
12           must be configured before Ethernet initialization and, the interrupt
13           priority should be the highest one.
14    *************************************************************************/
15
16    /* Configure Systick clock source as HCLK */
17    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
18
19    /* SystTick configuration: an interrupt every 10ms */
20    RCC_GetClocksFreq(&RCC_Clocks);
21    SysTick_Config(RCC_Clocks.HCLK_Frequency / 100);
22
23    /* Set Systick interrupt priority to 0*/
24    NVIC_SetPriority (SysTick_IRQn, 0);
25
26    /* Configure the GPIO ports for ethernet pins */
27    ETH_GPIO_Config();
28
29    /* Configure the Ethernet MAC/DMA */
30    ETH_MACDMA_Config();
31
32    /* Read PHY status register: Get Ethernet link status */
33    rc1 = ETH_ReadPHYRegister(LAN8720_PHY_ADDRESS, PHY_SR); // 原版的实现
```

```
    rc2 = ETH_ReadPHYRegister(LAN8720_PHY_ADDRESS, PHY_BSR); // 第三方的实现

    // 原版的判断不好使，第三方的判断好使，结合在一起用。有时间再去看LAN8720的数据表
    if ((rc1 & 0x01) || ((rc2 & 0x04) > 0))
    {
      EthStatus |= ETH_LINK_FLAG; // 在硬件连接没问题的情况下，必须到这里才正确.
    }

    // 注释掉LAN8720没有的寄存器操作
    /* Configure the PHY to generate an interrupt on change of link status */
    // Eth_Link_PHYITConfig(LAN8720_PHY_ADDRESS);

    /* Configure the EXTI for Ethernet link status. */
    // Eth_Link_EXTIConfig();
}
```

修改PHY地址宏

```
static void ETH_MACDMA_Config(void)
{
  /* Enable ETHERNET clock  */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_ETH_MAC | RCC_AHB1Periph_ETH_MAC_Tx |
                         RCC_AHB1Periph_ETH_MAC_Rx, ENABLE);

  /* Reset ETHERNET on AHB Bus */
  ETH_DeInit();

  /* Software reset */
  ETH_SoftwareReset();

  /* Wait for software reset */
  while (ETH_GetSoftwareResetStatus() == SET);

  /* ETHERNET Configuration --------------------------------------------------*/
  /* Call ETH_StructInit if you don't like to configure all ETH_InitStructure
parameter */
  ETH_StructInit(&ETH_InitStructure);

  /* Fill ETH_InitStructure parametrs */
  /*-------------------------   MAC   -----------------------------------*/
  ETH_InitStructure.ETH_AutoNegotiation = ETH_AutoNegotiation_Enable;
//  ETH_InitStructure.ETH_AutoNegotiation = ETH_AutoNegotiation_Disable;
//  ETH_InitStructure.ETH_Speed = ETH_Speed_10M;
//  ETH_InitStructure.ETH_Mode = ETH_Mode_FullDuplex;

  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_Perfect;
  ETH_InitStructure.ETH_UnicastFramesFilter = ETH_UnicastFramesFilter_Perfect;
#ifdef CHECKSUM_BY_HARDWARE
  ETH_InitStructure.ETH_ChecksumOffload = ETH_ChecksumOffload_Enable;
#endif

  /*-----------------------   DMA   -----------------------------------------*/
  /* When we use the Checksum offload feature, we need to enable the Store and Forward mode:
  the store and forward guarantee that a whole frame is stored in the FIFO, so the MAC can insert/verify the checksum,
  if the checksum is OK the DMA can handle the frame otherwise the frame is dropped */
  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 */
  // LAN8720_PHY_ADDRESS是我改名的宏
  EthStatus = ETH_Init(&ETH_InitStructure, LAN8720_PHY_ADDRESS);
}

```

只初始化RMII连接必要的GPIO

如果RMII必要的GPIO初始化错了，就会卡在如下实现

```
while (ETH_GetSoftwareResetStatus() == SET);
```

因为从寄存器读出的值不对了。

```
void ETH_GPIO_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Enable GPIOs clocks */
  // RMII所在的管脚时钟为PA, PC, PG
//  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB |
//                         RCC_AHB1Periph_GPIOC | RCC_AHB1Periph_GPIOI |
//                         RCC_AHB1Periph_GPIOG | RCC_AHB1Periph_GPIOH |
//                         RCC_AHB1Periph_GPIOF, ENABLE);

```

```
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOC |
RCC_AHB1Periph_GPIOG, ENABLE);

  /* Enable SYSCFG clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

  // 不需要时钟输出
  /* Configure MCO (PA8) */
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
//  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
//  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
//  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
//  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* MII/RMII Media interface selection --------------------------------------*/
#ifdef MII_MODE /* Mode MII with STM324xx-EVAL  */
 #ifdef PHY_CLOCK_MCO

  /* Output HSE clock (25MHz) on MCO pin (PA8) to clock the PHY */
  RCC_MCO1Config(RCC_MCO1Source_HSE, RCC_MCO1Div_1);
 #endif /* PHY_CLOCK_MCO */

  SYSCFG_ETH_MediaInterfaceConfig(SYSCFG_ETH_MediaInterface_MII);
#elif defined RMII_MODE  /* Mode RMII with STM324xx-EVAL */

  // 必须执行RMII接口的配置才行, 所以必须要定义RMII_MODE宏，注释掉MII_MODE宏
  SYSCFG_ETH_MediaInterfaceConfig(SYSCFG_ETH_MediaInterface_RMII);
#endif

// STM324xx-EVAL + MII + DP83848CVV
// Ethernet pins configuration
//
//        ETH_MDIO -------------------------> PA2
//        ETH_MDC --------------------------> PC1
//          ETH_PPS_OUT ----------------------> PB5
//          ETH_MII_CRS ----------------------> PH2
//          ETH_MII_COL ----------------------> PH3
//          ETH_MII_RX_ER --------------------> PI10
//          ETH_MII_RXD2 ---------------------> PH6
//          ETH_MII_RXD3 ---------------------> PH7
//          ETH_MII_TX_CLK -------------------> PC3
//          ETH_MII_TXD2 ---------------------> PC2
//          ETH_MII_TXD3 ---------------------> PB8
//        ETH_MII_RX_CLK/ETH_RMII_REF_CLK---> PA1
//        ETH_MII_RX_DV/ETH_RMII_CRS_DV ----> PA7
//        ETH_MII_RXD0/ETH_RMII_RXD0 -------> PC4
//        ETH_MII_RXD1/ETH_RMII_RXD1 -------> PC5
//        ETH_MII_TX_EN/ETH_RMII_TX_EN -----> PG11
//        ETH_MII_TXD0/ETH_RMII_TXD0 -------> PG13
//        ETH_MII_TXD1/ETH_RMII_TXD1 -------> PG14

//  /* Configure PA1, PA2 and PA7 */
//  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_7;
//  GPIO_Init(GPIOA, &GPIO_InitStructure);
//  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_ETH);
```

```
67   //  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_ETH);
68   //  GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_ETH);
69
70   //  /* Configure PB5 and PB8 */
71   //  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_8;
72   //  GPIO_Init(GPIOB, &GPIO_InitStructure);
73   //  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5, GPIO_AF_ETH);
74   //  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_ETH);
75
76   //  /* Configure PC1, PC2, PC3, PC4 and PC5 */
77   //  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
     GPIO_Pin_4 | GPIO_Pin_5;
78   //  GPIO_Init(GPIOC, &GPIO_InitStructure);
79   //  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1, GPIO_AF_ETH);
80   //  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_ETH);
81   //  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_ETH);
82   //  GPIO_PinAFConfig(GPIOC, GPIO_PinSource4, GPIO_AF_ETH);
83   //  GPIO_PinAFConfig(GPIOC, GPIO_PinSource5, GPIO_AF_ETH);
84   //
85   //  /* Configure PG11, PG14 and PG13 */
86   //  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14;
87   //  GPIO_Init(GPIOG, &GPIO_InitStructure);
88   //  GPIO_PinAFConfig(GPIOG, GPIO_PinSource11, GPIO_AF_ETH);
89   //  GPIO_PinAFConfig(GPIOG, GPIO_PinSource13, GPIO_AF_ETH);
90   //  GPIO_PinAFConfig(GPIOG, GPIO_PinSource14, GPIO_AF_ETH);
91
92   //  /* Configure PH2, PH3, PH6, PH7 */
93   //  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_6 |
     GPIO_Pin_7;
94   //  GPIO_Init(GPIOH, &GPIO_InitStructure);
95   //  GPIO_PinAFConfig(GPIOH, GPIO_PinSource2, GPIO_AF_ETH);
96   //  GPIO_PinAFConfig(GPIOH, GPIO_PinSource3, GPIO_AF_ETH);
97   //  GPIO_PinAFConfig(GPIOH, GPIO_PinSource6, GPIO_AF_ETH);
98   //  GPIO_PinAFConfig(GPIOH, GPIO_PinSource7, GPIO_AF_ETH);
99
100  //  /* Configure PI10 */
101  //  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
102  //  GPIO_Init(GPIOI, &GPIO_InitStructure);
103  //  GPIO_PinAFConfig(GPIOI, GPIO_PinSource10, GPIO_AF_ETH);
104
105  // STM32F407ZG + RMII + LAN8720
106  //
107  //        ETH_MDIO -------------------------> PA2
108  //        ETH_MDC --------------------------> PC1
109  //
110  //        ETH_MII_RX_CLK/ETH_RMII_REF_CLK---> PA1
111  //        ETH_MII_RX_DV/ETH_RMII_CRS_DV ----> PA7
112  //        ETH_MII_RXD0/ETH_RMII_RXD0 -------> PC4
113  //        ETH_MII_RXD1/ETH_RMII_RXD1 -------> PC5
114  //        ETH_MII_TX_EN/ETH_RMII_TX_EN -----> PG11
115  //        ETH_MII_TXD0/ETH_RMII_TXD0 -------> PG13
116  //        ETH_MII_TXD1/ETH_RMII_TXD1 -------> PG14
117  // 可以看出RMII接口比MII接口少了很多硬件连线
118
119  // 只能初始化必要的RMII接口的GPIO, 如果还初始化剩下的MII接口的GPIO, 程序就不对了。
120  // 管脚初始化
```

```
121     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
122     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
123       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
124     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
125 
126   //        ETH_MDIO -------------------------> PA2
127     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
128     GPIO_Init(GPIOA, &GPIO_InitStructure);
129     GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_ETH);
130 
131   //        ETH_MDC --------------------------> PC1
132     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
133     GPIO_Init(GPIOC, &GPIO_InitStructure);
134     GPIO_PinAFConfig(GPIOC, GPIO_PinSource1, GPIO_AF_ETH);
135 
136   //        ETH_MII_RX_CLK/ETH_RMII_REF_CLK---> PA1
137     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
138     GPIO_Init(GPIOA, &GPIO_InitStructure);
139     GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_ETH);
140 
141   //        ETH_MII_RX_DV/ETH_RMII_CRS_DV ----> PA7
142     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
143     GPIO_Init(GPIOA, &GPIO_InitStructure);
144     GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_ETH);
145 
146   //        ETH_MII_RXD0/ETH_RMII_RXD0 -------> PC4
147     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
148     GPIO_Init(GPIOC, &GPIO_InitStructure);
149     GPIO_PinAFConfig(GPIOC, GPIO_PinSource4, GPIO_AF_ETH);
150 
151   //        ETH_MII_RXD1/ETH_RMII_RXD1 -------> PC5
152     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
153     GPIO_Init(GPIOC, &GPIO_InitStructure);
154     GPIO_PinAFConfig(GPIOC, GPIO_PinSource5, GPIO_AF_ETH);
155 
156   //        ETH_MII_TX_EN/ETH_RMII_TX_EN -----> PG11
157     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
158     GPIO_Init(GPIOG, &GPIO_InitStructure);
159     GPIO_PinAFConfig(GPIOG, GPIO_PinSource11, GPIO_AF_ETH);
160 
161   //        ETH_MII_TXD0/ETH_RMII_TXD0 -------> PG13
162     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
163     GPIO_Init(GPIOG, &GPIO_InitStructure);
164     GPIO_PinAFConfig(GPIOG, GPIO_PinSource13, GPIO_AF_ETH);
165 
166   //        ETH_MII_TXD1/ETH_RMII_TXD1 -------> PG14
167     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
168     GPIO_Init(GPIOG, &GPIO_InitStructure);
169     GPIO_PinAFConfig(GPIOG, GPIO_PinSource14, GPIO_AF_ETH);
170   }
171 
172 
```

注释掉不要的MII实现的函数

```
// 这2个函数都不用的
/**
  * @brief  Configure the PHY to generate an interrupt on change of link status.
  * @param PHYAddress: external PHY address
  * @retval None
  */
//  uint32_t Eth_Link_PHYITConfig(uint16_t PHYAddress)
//  {
//    uint16_t tmpreg = 0;

//    /* Read MICR register */
//    tmpreg = ETH_ReadPHYRegister(PHYAddress, PHY_MICR);

//    /* Enable output interrupt events to signal via the INT pin */
//    tmpreg |= (uint16_t)(PHY_MICR_INT_EN | PHY_MICR_INT_OE);
//    if(!(ETH_WritePHYRegister(PHYAddress, PHY_MICR, tmpreg)))
//    {
//      /* Return ERROR in case of write timeout */
//      return ETH_ERROR;
//    }

//    /* Read MISR register */
//    tmpreg = ETH_ReadPHYRegister(PHYAddress, PHY_MISR);

//    /* Enable Interrupt on change of link status */
//    tmpreg |= (uint16_t)PHY_MISR_LINK_INT_EN;
//    if(!(ETH_WritePHYRegister(PHYAddress, PHY_MISR, tmpreg)))
//    {
//      /* Return ERROR in case of write timeout */
//      return ETH_ERROR;
//    }
//    /* Return SUCCESS */
//    return ETH_SUCCESS;
//  }

/**
  * @brief  EXTI configuration for Ethernet link status.
  * @param PHYAddress: external PHY address
  * @retval None
  */
  // RMII的硬件连接，没有这么连
//void Eth_Link_EXTIConfig(void)
//{
//  GPIO_InitTypeDef GPIO_InitStructure;
//  EXTI_InitTypeDef EXTI_InitStructure;
//  NVIC_InitTypeDef NVIC_InitStructure;

//  /* Enable the INT (PB14) Clock */
//  RCC_AHB1PeriphClockCmd(ETH_LINK_GPIO_CLK, ENABLE);
//  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

//  /* Configure INT pin as input */
//  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
```

```
//  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
//  GPIO_InitStructure.GPIO_Pin = ETH_LINK_PIN;
//  GPIO_Init(ETH_LINK_GPIO_PORT, &GPIO_InitStructure);

//  /* Connect EXTI Line to INT Pin */
//  SYSCFG_EXTILineConfig(ETH_LINK_EXTI_PORT_SOURCE, ETH_LINK_EXTI_PIN_SOURCE);

//  /* Configure EXTI line */
//  EXTI_InitStructure.EXTI_Line = ETH_LINK_EXTI_LINE;
//  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
//  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
//  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
//  EXTI_Init(&EXTI_InitStructure);

//  /* Enable and set the EXTI interrupt to priority 1*/
//  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
//  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
//  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
//  NVIC_Init(&NVIC_InitStructure);
//}


```

去掉不存在的MII寄存器读取

```
void Eth_Link_ITHandler(uint16_t PHYAddress)
{
  /* Check whether the link interrupt has occurred or not */
  // RMII接口没有PHY_MISR寄存器
  // if(((ETH_ReadPHYRegister(PHYAddress, PHY_MISR)) & PHY_LINK_STATUS) != 0)
  if (1)
  {
    if((ETH_ReadPHYRegister(PHYAddress, PHY_SR) & 1))
    {
      netif_set_link_up(&gnetif);
    }
    else
    {
      netif_set_link_down(&gnetif);
    }
  }
}

```

修正PHY地址宏改

```
void ETH_link_callback(struct netif *netif)
{
  __IO uint32_t timeout = 0;
 uint32_t tmpreg;
 uint16_t RegValue;
```

```
  struct ip_addr ipaddr;
  struct ip_addr netmask;
  struct ip_addr gw;
#ifndef USE_DHCP
  uint8_t iptab[4] = {0};
  uint8_t iptxt[20];
#endif /* USE_DHCP */

  /* Clear LCD */
  LCD_ClearLine(Line4);
  LCD_ClearLine(Line5);
  LCD_ClearLine(Line6);
  LCD_ClearLine(Line7);
  LCD_ClearLine(Line8);
  LCD_ClearLine(Line9);

  if(netif_is_link_up(netif))
  {
    /* Restart the auto-negotiation */
    if(ETH_InitStructure.ETH_AutoNegotiation != ETH_AutoNegotiation_Disable)
    {
      /* Reset Timeout counter */
      timeout = 0;

      /* Enable auto-negotiation */
      // 修正PHY地址宏为LAN8720_PHY_ADDRESS
      ETH_WritePHYRegister(LAN8720_PHY_ADDRESS, PHY_BCR, PHY_AutoNegotiation);

      /* Wait until the auto-negotiation will be completed */
      do
      {
        timeout++;
        // 修正PHY地址宏为LAN8720_PHY_ADDRESS
      } while (!(ETH_ReadPHYRegister(LAN8720_PHY_ADDRESS, PHY_BSR) &
PHY_AutoNego_Complete) && (timeout < (uint32_t)PHY_READ_TO));

      /* Reset Timeout counter */
      timeout = 0;

      /* Read the result of the auto-negotiation */
      // 修正PHY地址宏为LAN8720_PHY_ADDRESS
      RegValue = ETH_ReadPHYRegister(LAN8720_PHY_ADDRESS, PHY_SR);


```

修正PHY地址宏

\STM32F4x7_ETH_LwIP_V1.1.1\Project\Standalone\httpserver\src\stm32f4xx_it.c

```
void SysTick_Handler(void)
{
  /* Update the LocalTime by adding SYSTEMTICK_PERIOD_MS each SysTick interrupt
*/
  Time_Update();
```

```
}

/**
  * @brief  This function handles External line 10 interrupt request.
  * @param  None
  * @retval None
  */
void EXTI15_10_IRQHandler(void)
{
  if(EXTI_GetITStatus(ETH_LINK_EXTI_LINE) != RESET)
  {
    // // 修正PHY地址宏为LAN8720_PHY_ADDRESS
    Eth_Link_ITHandler(LAN8720_PHY_ADDRESS);
    /* Clear interrupt pending bit */
    EXTI_ClearITPendingBit(ETH_LINK_EXTI_LINE);
  }
}

```

PHY驱动实现

ST官方实现了RMII接口的PHY在lwip下的驱动实现, 不用改。
\STM32F4x7_ETH_LwIP_V1.1.1\Utilities\Third_Party\lwip-1.4.1\port\STM32F4x7\Standalone\ethernetif.c
必须实现的接口如下

```
// 底层的初始化，输入，输出
static void low_level_init(struct netif *netif);
static err_t low_level_output(struct netif *netif, struct pbuf *p);
static struct pbuf * low_level_input(struct netif *netif);

// 网卡层的初始化和输入
err_t ethernetif_input(struct netif *netif);
err_t ethernetif_init(struct netif *netif);

```

去掉警告

发现有几个警告，去掉

```
err_t
tcp_send_empty_ack(struct tcp_pcb *pcb)
{
  struct pbuf *p;
  // struct tcp_hdr *tcphdr; // 去掉警告
  u8_t optlen = 0;

#if LWIP_TCP_TIMESTAMPS
  if (pcb->flags & TF_TIMESTAMP) {
    optlen = LWIP_TCP_OPT_LENGTH(TF_SEG_OPTS_TS);
  }
#endif
```

```
13
14    p = tcp_output_alloc_header(pcb, optlen, 0, htonl(pcb->snd_nxt));
15    if (p == NULL) {
16      LWIP_DEBUGF(TCP_OUTPUT_DEBUG, ("tcp_output: (ACK) could not allocate
    pbuf\n"));
17      return ERR_BUF;
18    }
19    // tcphdr = (struct tcp_hdr *)p->payload; // 去掉警告
20    LWIP_DEBUGF(TCP_OUTPUT_DEBUG,
21                ("tcp_output: sending ACK for %"U32_F"\n", pcb->rcv_nxt));
22    /* remove ACK flags from the PCB, as we send an empty ACK now */
23    pcb->flags &= ~(TF_ACK_DELAY | TF_ACK_NOW);
24
```

```
1   void
2   tcp_keepalive(struct tcp_pcb *pcb)
3   {
4     struct pbuf *p;
5     // struct tcp_hdr *tcphdr; // 去掉警告
6
7     LWIP_DEBUGF(TCP_DEBUG, ("tcp_keepalive: sending KEEPALIVE probe to
    %"U16_F".%"U16_F".%"U16_F".%"U16_F"\n",
8                             ip4_addr1_16(&pcb->remote_ip), ip4_addr2_16(&pcb-
    >remote_ip),
9                             ip4_addr3_16(&pcb->remote_ip), ip4_addr4_16(&pcb-
    >remote_ip)));
10
11    LWIP_DEBUGF(TCP_DEBUG, ("tcp_keepalive: tcp_ticks %"U32_F"   pcb->tmr %"U32_F"
    pcb->keep_cnt_sent %"U16_F"\n",
12                            tcp_ticks, pcb->tmr, pcb->keep_cnt_sent));
13
14    p = tcp_output_alloc_header(pcb, 0, 0, htonl(pcb->snd_nxt - 1));
15    if(p == NULL) {
16      LWIP_DEBUGF(TCP_DEBUG,
17                  ("tcp_keepalive: could not allocate memory for pbuf\n"));
18      return;
19    }
20    // tcphdr = (struct tcp_hdr *)p->payload; // 去掉警告
21
22  #if CHECKSUM_GEN_TCP
23    tcphdr->chksum = inet_chksum_pseudo(p, &pcb->local_ip, &pcb->remote_ip,
24                                        IP_PROTO_TCP, p->tot_len);
25  #endif
26    TCP_STATS_INC(tcp.xmit);
27
28    /* Send output to IP */
29  #if LWIP_NETIF_HWADDRHINT
30    ip_output_hinted(p, &pcb->local_ip, &pcb->remote_ip, pcb->ttl, 0, IP_PROTO_TCP,
31      &(pcb->addr_hint));
32  #else /* LWIP_NETIF_HWADDRHINT*/
33    ip_output(p, &pcb->local_ip, &pcb->remote_ip, pcb->ttl, 0, IP_PROTO_TCP);
34  #endif /* LWIP_NETIF_HWADDRHINT*/
35
36    pbuf_free(p);
37
38    LWIP_DEBUGF(TCP_DEBUG, ("tcp_keepalive: seqno %"U32_F" ackno %"U32_F".\n",
```

```
39                              pcb->snd_nxt - 1, pcb->rcv_nxt));
40    }
41
```

试验效果

从原版上，经过以上修改后，编译通过，0错误0警告。
下载到板子上，跑起来。
从PC端ping板子，能ping通。

下面可以在这个修改完工程的基础上，将修改移植到自己出问题的工程中。网卡驱动 + lwip的运行就没问题了。
接下来，可以加入ucos, 将任务跑起来，在任务中，查询网络接口，收包，发包。