

W5500 移植教程

Version 1.0.0

目 录

1	W5500	3
1.1	简介	3
1.2	特点	3
1.3	目标应用	3
2	ioLibrary 库	4
2.1	简介	4
2.2	特点	4
2.3	库目录介绍	4
3	例程说明	7
3.1	介绍	7
3.2	STM32 HAL 库例程目录介绍	8
3.3	STM32 标准库例程目录介绍	9
4	移植示例	10
4.1	移植到 STM32 其他系列	10
4.2	移植到其他平台	10
4.3	移植实例	10
4.3.1	普冉 PY32F030K28U6TR	10
5	移植常见问题及解决方法	19
5.1	编译错误	19
5.2	硬件初始化适配	19
5.3	程序运行异常	19
6	文档历史信息	20

1 W5500

1.1 简介

W5500 是一款全硬件 TCP/IP 嵌入式以太网控制器，为嵌入式系统提供了更加 简易的互联网连接方案。W5500 集成了 TCP/IP 协议栈，10/100M 以太网数据链路层 (MAC) 及物理层 (PHY) ，使得用户使用单芯片就能够再他们的应用中拓展网络连接。

久经市场考验的 WIZnet 全硬件 TCP/IP 协议栈支持 TCP, UDP, IPv4, ICMP, ARP, IGMP 以及 PPPoE 协议。W5500 内嵌 32K 字节片上缓存以供以太网包处理。如果你使用 W5500，你只需要一些简单的 Socket 编程就能实现以太网应用。这将会比其他嵌入式以太网方案更加快捷、简便。用户可以同时使用 8 个硬件 Socket 独立通讯。

W5500 提供了 SPI (外设串行接口) 从而能够更加容易与外设 MCU 整合。而且，W5500 的使用了新的高效 SPI 协议支持 80MHz 速率，从而能够更好的实现高速网络通讯。为了减少系统能耗，W5500 提供了网络唤醒模式 (WOL) 及掉电模式 供客户选择使用。

1.2 特点

- 支持硬件 TCP/IP 协议：TCP, UDP, ICMP, IPv4, ARP, IGMP, PPPoE
- 支持 8 个独立端口 (Socket) 同时通讯
- 支持掉电模式
- 支持网络唤醒
- 支持高速串行外设接口 (SPI 模式 0, 3)
- 内部 32K 字节收发缓存
- 内嵌 10BaseT/100BaseTX 以太网物理层 (PHY)
- 支持自动协商 (10/100-Based 全双工/半双工)
- 不支持 IP 分片
- 3.3V 工作电压, I/O 信号口 5V 耐压
- LED 状态显示 (全双工/半双工, 网络连接, 网络速度, 活动状态)
- LQFP48 无铅封装 (7x7mm, 间距 0.5mm)

1.3 目标应用

W5500 适合于以下嵌入式应用：

- 家庭网络设备：机顶盒、个人录像机、数码媒体适配器
- 串行转以太网：门禁控制、LED 显示屏、无线 AP 继电器等
- 并行转以太网：POS/微型打印机、复印机
- USB 转以太网：存储设备、网络打印机
- GPIO 转以太网：家庭网络传感器
- 安全系统：数字录像机、网络摄像机、信息亭
- 工厂和楼宇自动化控制系统
- 医疗监测设备
- 嵌入式服务器

2 ioLibrary 库

2.1 简介

ioLibrary 的意思是 WIZnet 芯片的 “Internet 卸载库” 。它包括驱动程序和应用程序协议。该驱动程序 (ioLibrary) 可用于 W5500、W5300、W5200、W5100、W5100S 等 WIZnetTCP/IP 芯片的应用设计。

2.2 特点

ioLibrary 库的主要作用有：

1. **资源释放**: 主处理器无需处理底层 TCP/IP 协议栈 (如数据包分片、重传、校验等)，专注于业务逻辑。
2. **开发简化**: 开发者通过库提供的 API (如 Berkeley Socket) 直接调用网络功能，无需关心硬件细节。

2.3 库目录介绍

ioLibrary 库包含了 WIZnet 芯片驱动，应用示例，以及协议驱动三个部分，具体结构如下所示：

```
ioLibrary
├ Application
│   ├ loopback
│   |   ├ loopback.c      包含回环测试程序，验证芯片收发功能
│   |   └ loopback.h
│   └ multicast
│       ├ multicast.c    函数用于验证组播数据的收发功能
│       └ multicast.h
├ Ethernet          目录包含了针对不同 WIZnet 以太网芯片的驱动文件，用于与具体的芯片
  进行交互。
    ├ W5100
    |   ├ w5100.c
    |   └ w5100.h
    ├ W5100S
    |   ├ w5100s.c
    |   └ w5100s.h
    ├ W5200
    |   ├ w5200.c
    |   └ w5200.h
    ├ W5300
    |   ├ w5300.c
    |   └ w5300.h
    └ W5500
```

	↗ w5500.c
	↘ w5500.h
└	Internet 该目录包含了各种网络协议的实现代码，用于实现不同的网络应用。
	└ DHCP
	↗ dhcp.c 动态主机配置协议 (DHCP) 的实现文件，用于自动获取网络配置信息
	↘ dhcp.h
	└ DNS
	↗ dns.c 域名系统 (DNS) 的实现文件，用于将域名解析为对应的 IP 地址
	↘ dns.h
	└ FTPClient
	↗ ftpc.c FTP 客户端的实现文件，用于实现与 FTP 服务器的通信，完成文件的上传、下载等操作
	↗ ftpc.h
	↘ stdio_private.h
	└ FTPServer
	↗ ftpd.c FTP 服务器的实现文件，用于搭建一个简单的 FTP 服务器，允许客户端进行文件的上传和下载。
	↗ ftpd.h
	↗ REAME.md
	↘ stdio_private.h
	└ httpServer
	↗ httpParser.c HTTP 解析器的实现文件，用于解析 HTTP 请求消息。
	↗ httpParser.h
	↗ httpServer.c HTTP 服务器的实现文件，用于搭建一个简单的 HTTP 服务器，处理客户端的 HTTP 请求。
	↗ httpServer.h
	↗ httpUtil.c HTTP 相关的工具函数实现文件，提供一些辅助功能，如处理 HTTP 响应头、状态码等。
	↘ httpUtil.h
	└ MQTT
	↗ MQTTPacket MQTT 协议数据包处理相关的代码目录，包含了 MQTT 协议数据包的编码和解码等功能。
	↗ mqtt_interface.c MQTT 接口的实现文件，提供了与 MQTT 协议交互的接口函数。
	↗ mqtt_interface.h
	↗ MQTTClient.c MQTT 客户端的实现文件，用于实现与 MQTT 服务器的通信，完成消息的发布和订阅等操作。
	↘ MQTTClient.h
	└ SNMP
	↗ tools 包含 SNMP 协议相关的工具代码目录。
	↗ snmp.c 简单网络管理协议 (SNMP) 的实现文件，用于实现 SNMP 协议的基本功能，如获取设备信息等。

```
| └ snmp.h
| └ snmp_custom.c    自定义的 SNMP 功能实现文件，是对 SNMP 协议进行了一些扩展或
定制
|   └ snmp_custom.h
└ SNTP
| └ sntp.c          简单网络时间协议（SNTP）的实现文件，用于从 SNTP 服务器获取准确
的时间信息。
|   └ sntp.h
└ TFTP
  └ netutil.c    网络工具函数的实现文件，提供一些与网络通信相关的辅助功能
  └ netutil.h
  └ tftp.c        简单文件传输协议（TFTP）的实现文件，用于实现文件的简单传输功能。
  └ tftp.h
```

3 例程说明

3.1 介绍

W5500 芯片的例程目前有 STM32 HAL 库和 STM32 标准库两个平台下的 30 个例程，基本涵盖大部分以太网协议，方便客户参考和验证。

例程链接：[WIZnet HK \(wiznet-hk\) - Gitee.com](#)

例程如下：

- 1.Network_install：标准驱动例程；
- 2.DHCP：通过 DHCP 方式从服务器获取网络配置信息；
- 3.TCP_Client：以 TCP 客户端的方式和服务器进行数据回环测试；
- 4.TCP_Server：作为 TCP 服务器，回环客户端发来的数据；
- 5.UDP：以 UDP 方式回环接收到的数据；
- 6.UDP_Multicast：以 UDP 组播方式进行通信；
- 7.DNS：通过 DNS 解析 wiznet.io 的 IP 地址；
- 8.HTTP_Client：HTTP 发起请求示例；
- 9.HTTP_Server：作为 HTTP 服务器，处理客户端请求；
- 10.SNTP：获取网络时间；
- 11.SMTP：通过 SMTP 协议快速发送邮件；
- 12.NetBIOS：通过 NetBIOS 协议，让其他用户可以以类似域名的格式发起 ping 请求；
- 13.UPnP：通过 UPnP 协议快速控制设备；
- 14.TFTP：以客户端身份连接 TFTP 服务器下载文件；
- 15.SNMP：通过 SNMP 协议管理网络设备；
- 16.PING：IPRAW 模式 ping 设备；
- 17.ARPs：硬件 ARP 示例；
- 18.FTP_Server：FTP 服务器模式示例，其他设备可以通过访问服务器下载文件；
- 19.FTP_Client：FTP 客户端模式示例，可以访问 FTP 服务器下载文件；
- 20.WOL：通过网络唤醒设备；
- 21.PHY_Mode_Config：软件控制 PHY 模式；
- 22.MQTT&Aliyun：通过 MQTT 协议连接阿里云平台并实现数据交互；
- 23.MQTT&OneNET：通过 MQTT 协议连接 OneNET 平台并实现数据交互；
- 24.TCP_Client_Multi_socket：开启全部 socket 连接 TCP 服务器；
- 25.TCP_Server_Multi_socket：作为 TCP 服务器，允许多个设备连接进行数据通信；
- 26.Upper_computer_search_and_config：通过上位机修改配置；
- 27.interrupt：中断回环数据示例；
- 28.Ethernet_Rate_Test：以太网测速示例；
- 29.Modbus_TCP_Server：Modbus TCP 示例；
- 30.HTTP_Server&NetBIOS：DHCP+HTTP Server+NetBIOS 示例；

3.2 STM32 HAL 库例程目录介绍

STM32 HAL 库例程目录如图 1 所示，包含 4 个主要文件夹：

Core：通常存放项目核心代码，包含核心功能的头文件 (.h) 和源文件 (.c)。

Drivers：设备驱动目录，用于存放硬件驱动程序，实现对芯片外设或外部设备的控制。

MDK-ARM：与 Keil MDK 开发环境相关，一般包含工程文件 (.uvprojx 等)，用于管理项目编译、配置等。

User：用户自定义文件夹，用于存放用户编写的代码，包含以太网驱动库、用户主程序、以太网接口以及硬件平台适配以太网目录。

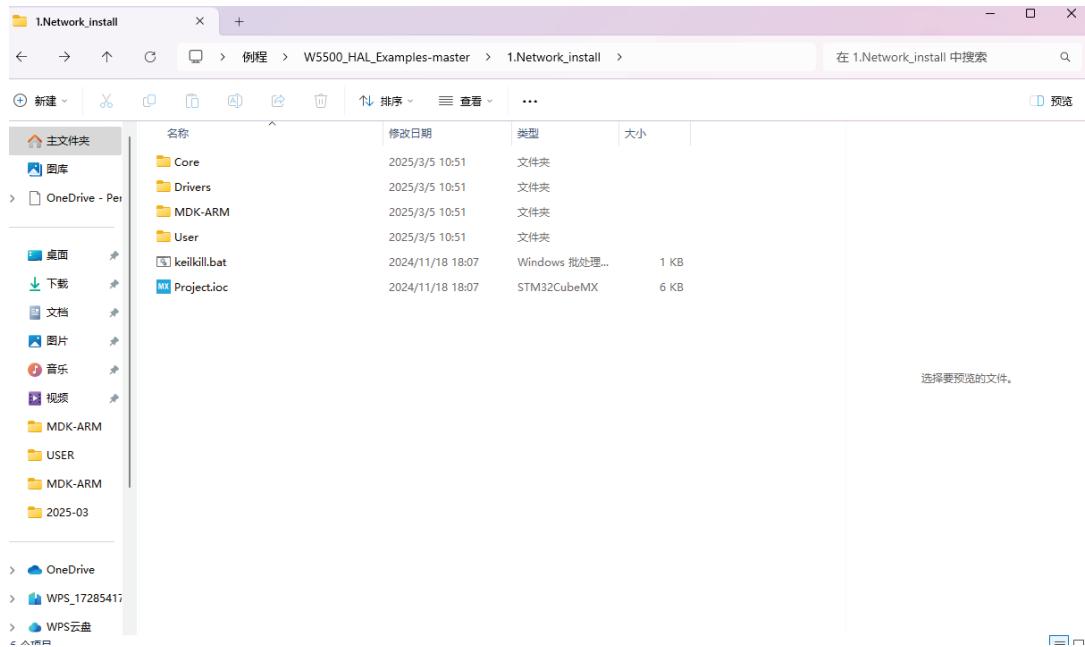


图 1 STM32 HAL 库例程目录示例

示例工程结构如下：

Project

```

|   |- Core
|   |   |- Inc      头文件目录
|   |   |- Src      源文件目录
|   |- Drivers    设备驱动目录
|   |- MDK-ARM   Keil 工程目录
|   |   |- Project
|   |- User       用户文件夹
|   |   |- ioLibrary_Driver 以太网驱动库
|   |   |- user_main    用户主程序
|   |   |- wiz_interface 以太网接口
|   |   |- wiz_platform   硬件平台适配以太网目录

```

3.3 STM32 标准库例程目录介绍

STM32 标准库目录如图 2 所示：

ioLibrary_Driver: 存放以太网驱动库文件，用于实现网络功能的底层驱动代码。

Libraries: 存储项目依赖的基础库文件，stm32f10x 驱动库。

MDK-ARM: Keil MDK 开发环境的工程目录，内含项目工程文件（如 .uvprojx）、编译配置、输出设置等，用于管理项目构建。

User: 用户自定义代码文件夹，用于存放主程序、业务逻辑代码或硬件平台适配代码，是开发者主要编写代码的目录，里面包含以太网接口以及硬件平台适配以太网目录。

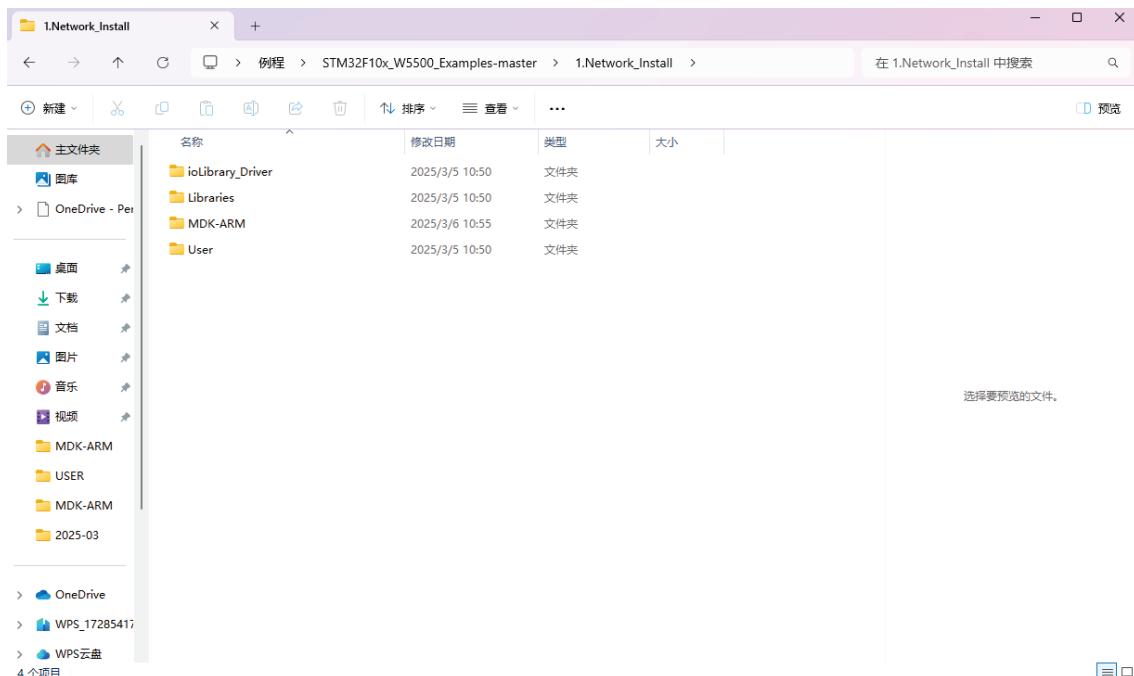


图 2 STM32 标准库例程目录示例

示例工程结构如下：

Project

- | | └── ioLibrary_Driver 以太网驱动库
- | | └── Libraries stm32 驱动库
- | | └── Drivers 设备驱动目录
- | | └── MDK-ARM Keil 工程目录
- | | └── User 用户文件夹
- | | └── wiz_interface 以太网接口
- | | └── wiz_platform 硬件平台适配以太网目录

4 移植示例

4.1 移植到 STM32 其他系列

如果需要移植例程到 STM32 的其他系列，建议从 HAL 库开始移植，可以按照以下步骤进行操作：

1. 打开 STM32CubeMX，选择目标芯片
2. 按照示例工程配置同样的外设以及 GPIO

RSTn	PD8
INTn	PD9
SCSn	PD7
USART1_TX	PA9
USART1_RX	PA10
SPI2_SCK	PB13
SPI2_MISO	PB14
SPI2_MOSI	PB15
3. 配置定时器以及中断
4. 根据自己需求配置时钟树
5. 生成工程 HAL 库工程
6. 将例程中的 ioLibrary 库和 User 文件夹添加到生成的工程中
7. 在新工程中添加相应的文件以及路径
8. 在生成工程的 main.c 文件中调用 user_run() 函数

4.2 移植到其他平台

如果需要移植到 STM32 其他系列的标准库工程或者是其他芯片平台上，建议从 STM32 标准库例程进行移植，可以按照以下步骤操作：

1. 新建目标系列的工程
2. 将示例中的 ioLibrary 库和 User 文件夹复制到新平台文件夹中
3. 在新平台工程添加对应的文件和路径
4. 在 wiz_platform.c 文件中适配新平台驱动部分。例如：
 - 1) 串口相关函数
 - 2) SPI 相关函数
 - 3) GPIO 相关函数
 - 4) 定时器相关函数
5. 将例程中的 main.c 移植到新平台的 main.c 中

4.3 移植实例

4.3.1 普冉 PY32F030K28U6TR

- (1) 下载普冉芯片系列的官方示例工程进创建。
- (2) 把 ioLibrary 库以及 User 文件夹添加到工程文件夹下，并添加到工程中，如图 3 所示



The screenshot shows the uVision IDE interface with the following details:

- Project Tree:** Shows the project structure under "PY32F030 Project".
 - Common: startup_py32F0xx.s, system_py32F0xx.c
 - Application/User: main.c, py32F0xx_itc, py32F0xx_hal_msp.c, Bsp_Uart.c, spi.c, gpio.c, tim.c
 - PY32F0xx HAL Driver:
 - Ethernet: socket.c, w5500.c, wizchip_conf.c
 - Internet: dhcp.c
 - User: user_main.c, wiz_interface.c, wiz_platform.c, loopback.c
 - CMSIS
- Code Editor:** Displays the content of `user_main.c`. The code includes hardware pin definitions for the Wizchip, network configuration (MAC, IP, Subnet Mask, DNS), and a user run program that initializes the Wizchip, sets up the network, and starts a loopback UDP server on port 8000.

图 3 工程添加文件

(3) 添加路径

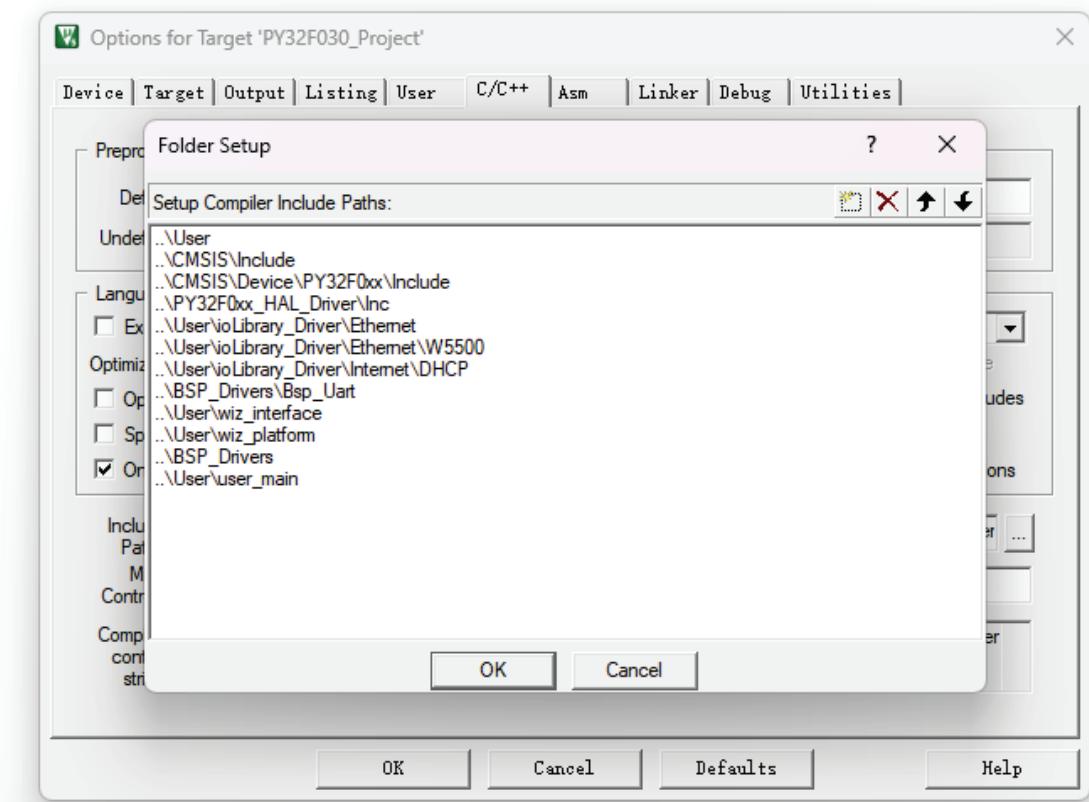


图 4 工程添加路径

(4) 适配 wiz_platform.c 文件

SPI 引脚连接

```

1. wizchip_SCS    ---> PY32_GPIOA3
2. wizchip_SCLK   ---> PY32_GPIOB3
3. wizchip_MISO   ---> PY32_GPIOB4
4. wizchip_MOSI   ---> PY32_GPIOB5
5. wizchip_RESET---> PY32_GPIOA13
6. wizchip_INT    ---> PY32_GPIOA15

```

SPI 初始化

```

1. SPI_HandleTypeDef hspi1;
2. void MX_SPI1_Init(void)
3. {
4.     hspi1.Instance = SPI1;
5.     hspi1.Init.Mode = SPI_MODE_MASTER;
6.     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
7.     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
8.     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
9.     hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
10.    hspi1.Init.NSS = SPI_NSS_SOFT;
11.    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
12.    hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
13.
14.    if (HAL_SPI_Init(&hspi1) != HAL_OK)
15.    {
16.        Error_Handler();
17.    }
18. }
19. void MX_BBBGPIO_Init(void)
20. {
21.     GPIO_InitTypeDef GPIO_InitStruct = {0};
22.     __HAL_RCC_GPIOB_CLK_ENABLE();
23.     __HAL_RCC_SPI1_CLK_ENABLE();
24.     GPIO_InitStruct.Pin = GPIO_PIN_3;
25.     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
26.     GPIO_InitStruct.Pull = GPIO_NOPULL;
27.     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
28.     GPIO_InitStruct.Alternate = GPIO_AF0_SPI1;
29.     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
30.     GPIO_InitStruct.Pin = GPIO_PIN_4;
31.     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
32.     GPIO_InitStruct.Pull = GPIO_PULLUP;
33.     GPIO_InitStruct.Alternate = GPIO_AF0_SPI1;
34.     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
35.     GPIO_InitStruct.Pin = GPIO_PIN_5;
36.     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
37.     GPIO_InitStruct.Pull = GPIO_NOPULL;
38.     GPIO_InitStruct.Alternate = GPIO_AF0_SPI1;
39.     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
40. }

```

USART 初始化

```

1.
2. USART_HandleTypeDef Uart1_Handle;
3. void DEBUG_USART_Config(uint32_t BaudRate)
4. {
5.     Uart1_Handle.Instance = USART1;
6.     Uart1_Handle.Init.BaudRate = BaudRate;
7.     Uart1_Handle.Init.WordLength = UART_WORDLENGTH_8B;
8.     Uart1_Handle.Init.StopBits = UART_STOPBITS_1;
9.     Uart1_Handle.Init.Parity = UART_PARITY_NONE;
10.    Uart1_Handle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
11.    Uart1_Handle.Init.Mode = UART_MODE_TX_RX;
12.
13.    HAL_UART_Init(&Uart1_Handle);
14.
15.    __HAL_UART_ENABLE_IT(&Uart1_Handle, UART_IT_RXNE);
16. }
17. void HAL_UART_MspInit(UART_HandleTypeDef *UartHandle)
18. {

```

```

19.    if(UartHandle->Instance == USART1)
20.    {
21.        GPIO_InitTypeDef      Uart_GPIO_InitConfig;
22.
23.        DEBUG_UART_Rx_GPIO_CLK_ENABLE();
24.        DEBUG_UART_Tx_GPIO_CLK_ENABLE();
25.        __HAL_RCC_USART1_CLK_ENABLE();
26.        Uart_GPIO_InitConfig.Pin      = DEBUG_UART_Tx_GPIO_PIN;
27.        Uart_GPIO_InitConfig.Mode   = GPIO_MODE_AF_PP;
28.        Uart_GPIO_InitConfig.Pull   = GPIO_PULLUP;
29.        Uart_GPIO_InitConfig.Speed  = GPIO_SPEED_FREQ_HIGH;
30.        Uart_GPIO_InitConfig.Alternate = GPIO_AF0_USART1;
31.        HAL_GPIO_Init(DEBUG_UART_Tx_GPIO_PORT, &Uart_GPIO_InitConfig);
32.        Uart_GPIO_InitConfig.Pin      = DEBUG_UART_Rx_GPIO_PIN;
33.        Uart_GPIO_InitConfig.Mode   = GPIO_MODE_AF_PP;
34.        Uart_GPIO_InitConfig.Pull   = GPIO_PULLUP;
35.        Uart_GPIO_InitConfig.Speed  = GPIO_SPEED_FREQ_HIGH;
36.        Uart_GPIO_InitConfig.Alternate = GPIO_AF0_USART1;
37.        HAL_GPIO_Init(DEBUG_UART_Rx_GPIO_PORT, &Uart_GPIO_InitConfig);
38.        HAL_NVIC_SetPriority(USART1_IRQn, 0, 1);
39.        HAL_NVIC_EnableIRQ(USART1_IRQn);
40.    }
41.
42. USART1_IRQHandler(void)
43.
44.    if(__HAL_UART_GET_IT_SOURCE(&Uart1_Handle, UART_IT_RXNE))
45.    {
46.        uint8_t ch = READ_REG(Uart1_Handle.Instance->DR);
47.        WRITE_REG(Uart1_Handle.Instance->DR, ch);
48.    }
49. }
50. void Usart_SendString(uint8_t *str)
51. {
52.     unsigned int k=0;
53.     do
54.     {
55.         HAL_UART_Transmit(&Uart1_Handle, (uint8_t *)str + k) ,1,1000);
56.         k++;
57.     } while(*((str + k))!=\'0\');
58. }
59.
60. int fputc(int ch, FILE *f)
61. {
62.
63.     HAL_UART_Transmit(&Uart1_Handle, (uint8_t *)&ch, 1, 1000);
64.     return (ch);
65. }
66.
67. int fgetc(FILE *f)
68. {
69.     int ch;
70.     HAL_UART_Receive(&Uart1_Handle, (uint8_t *)&ch, 1, 0xffff);
71.     return (ch);
72. }
73. #elif defined(__ICCARM__)
74.
75. int putchar(int ch)
76. {
77.     HAL_UART_Transmit(&DebugUartHandle, (uint8_t *)&ch, 1, 1000);
78.     return (ch);
79. }
80. #elif defined(__GNUC__)
81.
82. int __io_putchar(int ch)
83. {
84.
85.     HAL_UART_Transmit(&DebugUartHandle, (uint8_t *)&ch, 1, 1000);
86.     return ch;
87. }
88. int _write(int file, char *ptr, int len)
89. {
90.     int DataIdx;

```

```

91. for (DataIdx=0; DataIdx<len; DataIdx++)
92. {
93.     __io_putchar(*ptr++);
94. }
95. return len;
96. }
97. #endif
98.

```

GPIO 初始化以及引脚修改

```

1. #define RSTn_Pin GPIO_PIN_11
2. #define RSTn_GPIO_Port GPIOA
3. #define INTn_Pin GPIO_PIN_15
4. #define INTn_GPIO_Port GPIOA
5. #define SCSn_Pin GPIO_PIN_3
6. #define SCSn_GPIO_Port GPIOA

```

GPIO 初始化

```

1.
2. void MX_GPIO_Init(void)
3. {
4.     GPIO_InitTypeDef GPIO_InitStruct = {0};
5.     /* GPIO Ports Clock Enable */
6.     __HAL_RCC_GPIOB_CLK_ENABLE();
7.     __HAL_RCC_GPIOA_CLK_ENABLE();
8.     /*Configure GPIO pin Output Level*/
9.     HAL_GPIO_WritePin(GPIOA, RSTn_Pin|SCSn_Pin, GPIO_PIN_SET);
10.    /*Configure GPIO pins : PDPin PDPin */
11.    GPIO_InitStruct.Pin = RSTn_Pin|SCSn_Pin;
12.    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
13.    GPIO_InitStruct.Pull = GPIO_PULLUP;
14.    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
15.    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
16.    /*Configure GPIO pin : PtPin */
17.    GPIO_InitStruct.Pin = INTn_Pin;
18.    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
19.    GPIO_InitStruct.Pull = GPIO_NOPULL;
20.    HAL_GPIO_Init(INTn_GPIO_Port, &GPIO_InitStruct);
21. }

```

定时器初始化以及中断配置

```

1. /**
2.  * @file  tim.c
3.  * @brief This file provides code for the configuration of TIM3.
4.  */
5.
6.
7.
8. #include "tim.h"
9.
10. TIM_HandleTypeDef htim3;
11.
12. void MX_TIM3_Init(void)
13. {
14.     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
15.     TIM_MasterConfigTypeDef sMasterConfig = {0};
16.     htim3.Instance = TIM3;
17.     htim3.Init.Period      = 1000 - 1;
18.     htim3.Init.Prescaler   = 48 - 1;
19.     htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
20.     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
21.     htim3.Init.RepetitionCounter = 1 - 1;
22.     htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
23.     HAL_TIM_Base_Init(&htim3);
24.     HAL_TIM_Base_Start_IT(&htim3);
25.     if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
26.     {
27.         Error_Handler();
28.     }
29.     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
30.     if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
31.
32.

```

```

33. {
34.     Error_Handler();
35. }
36. sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
37. sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
38. if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
39. {
40.     Error_Handler();
41. }
42. }
43. void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
44. {
45.     if(tim_baseHandle->Instance==TIM3)
46.     {
47.         __HAL_RCC_TIM3_CLK_ENABLE();
48.     }
49.     HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);
50.     HAL_NVIC_EnableIRQ(TIM3_IRQn);
51. }
52. }
53.
54. void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* tim_baseHandle)
55. {
56.     if(tim_baseHandle->Instance==TIM3)
57.     {
58.         __HAL_RCC_TIM3_CLK_DISABLE();
59.     }
60.     HAL_NVIC_DisableIRQ(TIM3_IRQn);
61. }
62. }
63. void TIM3_IRQHandler(void)
64. {
65.     HAL_TIM_IRQHandler(&htim3);
66. }

```

SPI 读写函数适配

```

1. /**
2. * @brief SPI write 1 byte to wizchip
3. * @param dat:1 byte data
4. * @return none
5. */
6. void wizchip_write_byte(uint8_t dat)
7. {
8.     HAL_SPI_Transmit(&hspi1, &dat, 1, 0xffff);
9. }
10.
11. /**
12. * @brief SPI read 1 byte from wizchip
13. * @param none
14. * @return 1 byte data
15. */
16. uint8_t wizchip_read_byte(void)
17. {
18.     uint8_t dat;
19.     HAL_SPI_Receive(&hspi1, &dat, 1, 0xffff);
20.     return dat;
21. }
22.
23. /**
24. * @brief SPI write buff from wizchip
25. * @param buff:write buff
26. * @param len:write len
27. * @return none
28. */
29. void wizchip_write_buff(uint8_t *buf, uint16_t len)
30. {
31.     HAL_SPI_Transmit(&hspi1, buf, len, 0xffff);
32. }
33.
34. /**
35. * @brief SPI read buff from wizchip
36. * @param buff:read buff

```

```
37. * @param len:read len
38. * @return none
39. */
40. void wizchip_read_buff(uint8_t *buf, uint16_t len)
41. {
42.     HAL_SPI_Receive(&hspi1, buf, len, 0xffff);
43. }
44.
```

定时器函数适配

```
1. /**
2. * @brief Hardware Platform Timer Interrupt Callback Function
3.
4. */
5. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
6. {
7.     if (htim->Instance == TIM3)
8.     {
9.         wiz_timer_handler();
10.    }
11. }
12.
13. /**
14. * @brief Turn on wiz timer interrupt
15. * @param none
16. * @return none
17. */
18. void wiz_tim_irq_enable(void)
19. {
20.
21.
22.     HAL_TIM_Base_Start_IT(&htim3);
23.
24. }
25.
26. /**
27. * @brief Turn off wiz timer interrupt
28. * @param none
29. * @return none
30. */
31. void wiz_tim_irq_disable(void)
32. {
33.     HAL_TIM_Base_Stop_IT(&htim3);
34. }
```



(5) 适配 main.c 文件

C:\Users\Desktop\PY32F030_UMDK-ARM\Project\upy32f030 -> vVision [Non-Commercial Use License]

Edit View Project File Debug Peripherals Tools SVCS Windows Help

Project PY32F030_Project

PY32F030_Project

Comm

system,py32f030xx.c

system,py32f030xx.c

Application/User

py32f030.h

py32f030_a.c

py32f030_h.msp.c

Bsp_Uart.c

spi.c

gpio.c

tim.c

PY32F030_HAL_Driver

Ethernet

w5500.c

w5500.h

internetchip_conf.c

internet

dhcp.c

User

user_main.c

wiz_interface.c

wiz_platform.c

loopback.c

CMSIS

main.c

```
22 /* Includes */
23 #include "main.h"
24 #include "spi.h"
25 #include "tim.h"
26 #include "gpio.h"
27 #include "HAL.h"
28 #include "user_main.h"
29 /* Private define */
30 /* Private user code */
31 /* Private macro */
32 /* Private function prototypes */
33 static void APP_SystemClockConfig(void);
34
35 /**
36 * @brief 应用程序入口函数。
37 */
38 int main(void)
39 {
40     HAL_Init();
41     MX_BB8BGPIO_Init();
42     APP_SystemClockConfig();
43     DEMUX_USART_Config((115200));
44     MX_GPIO_Init();
45     MX_SPI1_Init();
46     MX_TIM1_Init();
47     user_main();
48     while (1)
49     {
50     }
51 }
52
53 static void APP_SystemClockConfig(void)
54 {
55     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
56     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
57
58     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE | RCC_OSCILLATORTYPE_HSI | RCC_OSCILLATORTYPE_LSI | RCC_OSCILLATORTYPE_LSE; /* 选择振荡器HSE,HSI,LSI,LSE */
59     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
60     RCC_OscInitStruct.HSICalibrationValue = 0;
61
62     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
63     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
64     RCC_OscInitStruct.PLL.PLLM = 4;
65     RCC_OscInitStruct.PLL.PLLN = 32;
66     RCC_OscInitStruct.PLL.PLLP = 2;
67     RCC_OscInitStruct.PLL.PLLQ = 2;
68     RCC_OscInitStruct.PLL.PLLR = 2;
69     RCC_OscInitStruct.PLL.PLLFreq = RCC_HSE_16MHz;
70
71     RCC_OscInitStruct.PLL.PLL.PLLLock = RCC_PLL_LOCKED;
72     RCC_OscInitStruct.PLL.PLL.PLLState = RCC_PLL_ON;
73
74     /* 配置振荡器 */
75     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
76     {
77         Error_Handler();
78     }
79
80     /* 时钟配置 */
81     RCC_OscInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_PCLK1; /* 选择配置时钟 HCLK, SYSCLK, PCLK1 */
82     RCC_OscInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK; /* 选择PLLCLK作为系统时钟 */
83     RCC_OscInitStruct.AHBClockDivider = RCC_SYSCLK_DIV1; /* AHB时钟 1分频 */
84     RCC_OscInitStruct.APB1ClockDivider = RCC_HCLK_DIV1; /* APB1时钟 1分频 */
85
86     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
```

图 5 适配 main.c 文件

(6) 烧录验证

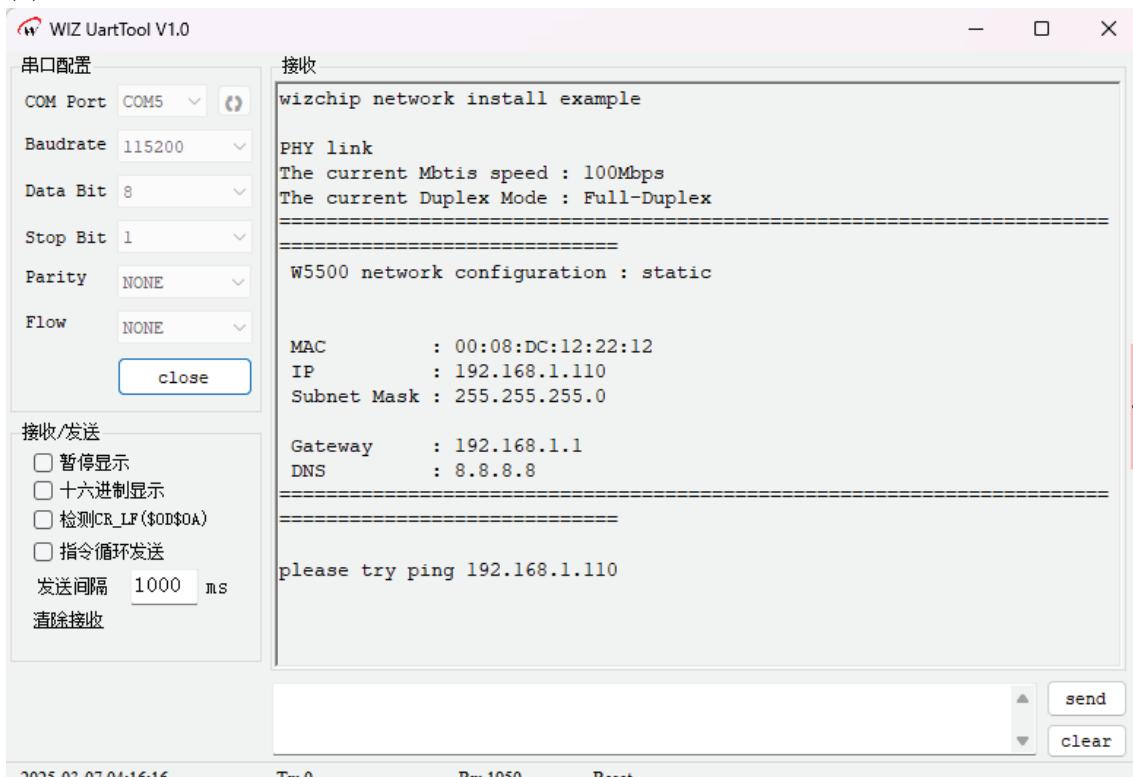


图 6 串口打印日志

```
C:\Windows\system32\cmd.exe + v
Microsoft Windows [版本 10.0.22631.4890]
(c) Microsoft Corporation。保留所有权利。

C:\Users\f>ping 192.168.1.110

正在 Ping 192.168.1.110 具有 32 字节的数据：
来自 192.168.1.110 的回复：字节=32 时间<1ms TTL=128

192.168.1.110 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

C:\Users\f>|
```

图 7 ping 验证结果

5 移植常见问题及解决方法

5.1 编译错误

1. **头文件找不到**: 检查头文件路径是否正确设置，确保所需的头文件在指定路径下。
2. **函数重定义**: 可能是源程序中包含了多个相同功能的函数定义，或者目标单片机 SDK 中的函数与源程序中的函数冲突。需检查代码，删除重复定义，或修改函数名以避免冲突。

5.2 硬件初始化适配

1. **引脚配置错误**: 仔细核对硬件连接和引脚配置代码，确保引脚功能设置正确。
2. **电源问题**: 检查开发板的电源供应是否正常，某些外设可能需要特定的电源电压才能正常工作。

5.3 程序运行异常

1. **中断处理错误**: 检查中断向量表是否正确配置，中断服务函数是否编写正确。可能是中断优先级设置不当，导致中断响应异常。
2. **内存溢出**: 查看程序中是否存在数组越界、动态内存分配不合理等情况，导致内存溢出。可通过优化代码、合理分配内存解决。

6 文档历史信息

版本	日期	描述
Ver. 1.0.0	2025-03-31	第一版发布

版权声明

Copyright 2024 WIZnet H.K. Limited 版权所有
技术支持:support@wiznet.hk
销售 & 代理:sales@wiznet.hk
更多信息, 请登录:<https://www.wiznet.io>