使用 LwIP TCP/IP 栈，在 STM32Cube 上开发应用

前言

STM32F4x7/9xx 和 STM32F2x7xx 微控制器配有高质量 10/100 Mbit/s 以太网外设，支持媒体独立接口 （MII）和缩减的媒体独立接口 （RMII），以便与物理层 （PHY）接口。

当使用以太网通信接口时，会使用 TCP/IP 软件协议栈以实现局域网或者广域网中的通信。

本用户手册说明了怎样使用 STM32CubeF2 和 STM32CubeF4 HAL 驱动程序，将一个免费中间件 TCP/IP 栈分别集成到基于 STM32F2x7xx 和 STM32F4x7/9xx 微控制器的嵌入式应用（请参考*第 1 章节*以获得 STM32Cube 的详细信息）。该中间件 TCP/IP 栈为 LwIP （轻量级 IP），专为嵌入式应用开发的开源协议栈。

对于每款评估 板，此包都包含了九个运行于 LwIP 栈之上的应用：

- 基于 Raw API，运行于独立模式 （没有 RTOS）的应用：
  - Web 服务器
  - TFTP 服务器
  - TCP 回响客户端应用
  - TCP 回响服务器应用
  - UDP 回响客户端应用
  - UDP 回响服务器应用
- 运行于 FreeRTOS 操作系统的应用：
  - 基于 netconn API 的 Web 服务器
  - 基于 socket API 的 Web 服务器
  - 基于 netconn API 的 TCP/UDP 回响服务器应用。

注： *在本文中， STM32Cube™ 指的是 STM32CubeF2 和 STM32CubeF4， STM32F4xx 指的是 STM32F4x7xx 和 STM32F4x9xx 微控制器， STM322xx-EVAL 和 STM324xx-EVAL 指的是 STM3221x-EVAL、 STM324xG-EVAL 和 STM324x9I-EVAL 评估板。*

本文提供的截屏和文件名对应的是运行于 STM32F4 微控制器上的应用样例。然而，它们也适用于 STM32F2x7xx。

目录

表格索引

图片索引

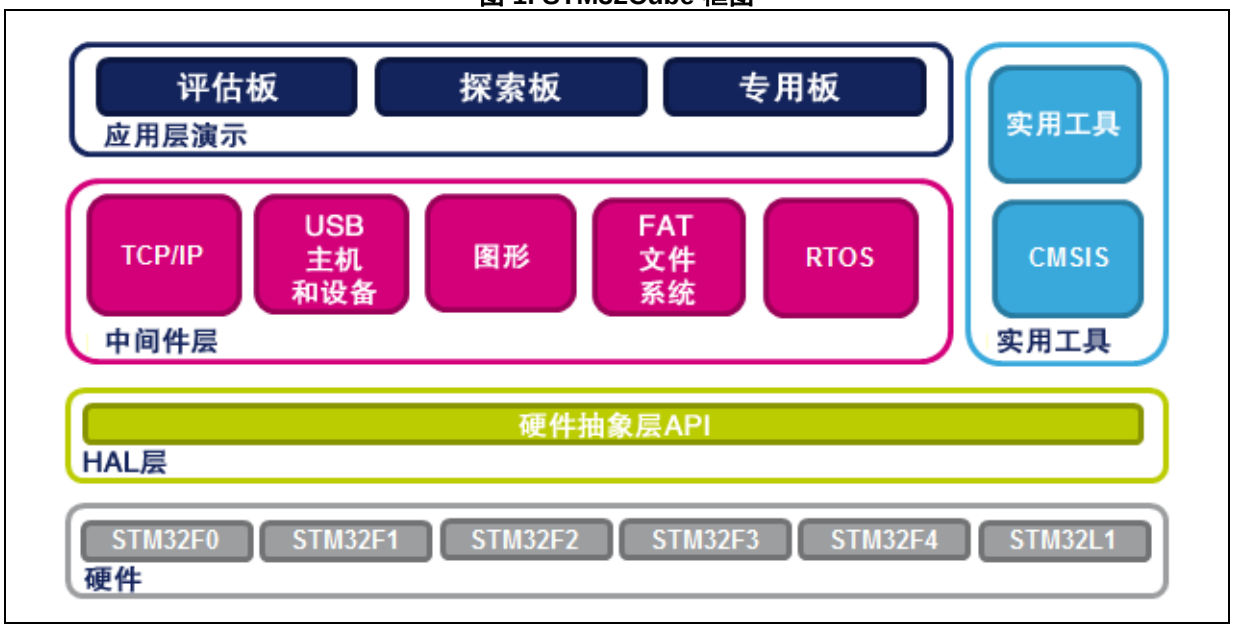
1 STM32Cube™ 概述

STM32Cube™ 计划源自意法半导体，旨在通过减少开发的工作量、时间与成本，使开发者受益。 STM32Cube 涵盖 STM32 产品组合。

STM32Cube 1.x 版包括：

- 图形软件配置工具 STM32CubeMX，可通过图形向导生成初始化 C 代码。
- 针对每个系列提供综合的嵌入式软件平台 （例如 STM32CubeF2 用于 STM32F2 系列，STM32CubeF4 用于 STM32F4 系列）
  - STM32 抽象层嵌入式软件 STM32Cube HAL，确保在 STM32 各个产品之间实现最大限度的可移植性
  - 一套一致的中间件，比如 RTOS、 USB、 TCP/IP、图形
  - 所有嵌入式软件实用工具均配备一套完整的示例。

图 1. STM32Cube 框图



2 LwIP TCP/IP 栈描述

2.1 栈特性

LwIP 为免费 TCP/IP 栈，由 Adam Dunkels 在瑞典计算机科学院 （SICS）开发，由修正的 BSD 许可授权。

LwIP TCP/IP 实现的侧重点为在全面保持 TCP/IP 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）

LwIP 具有三种应用编程接口 （API）：

- **Raw API** 为原始的 LwIP API。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。
- **Netconn API** 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）。 Netconn API 支持多线程工作。
- **BSD Socket API**：类似 Berkeley 的套接字 API （开发于 Netconn API 之上）

LwIP 栈的源代码可从 *http://savannah.nongnu.org* 下载。

2.2 授权

LwIP 由 BSD 许可证完成。下面是 LwIP 授权文档副本，它也包括在源代码中：

```
/*
 * 瑞典计算机科学院版权所有 （c） 2001-2004。
* 保留所有权利。
 *
 * 若要以源代码或二进制形式对其或使用，不管修改与否，
 * 都必须满足下述条件：
 *
 * 1. 对源代码重新发布时必须保留以上版权说明、
 *      此条件列表及下述免责声明。
```

```
* 2. 以二进制形式重新发布时必须将以上版权说明、
*      此条件列表及下述免责声明复制到文档
*      和 / 或其它一同发布的材料中。
* 3. 未经事先书面允许，不可使用作者姓名支持或宣传
*      由本软件衍生的产品。
*
* 此软件由其作者原样提供，不提供任何明示或暗示
* 保证，包括但不限于对
* 特定用途的适销性和适用性。在任何情况下，
* 作者对任何直接、间接、附带、特殊、
* 惩罚性或后果性损失 （包括但不限于，采购
* 替代商品或服务；使用、数据或利润损失；或业务
* 中断）。在任何赔偿理论下，不管是否存在
* 合同、严格责任，或民事侵权 （包括过失或其它），
* 都不能归咎于使用此软件，即使已被告知
* 此类损害的可能性。
*
* 此文件为 lwIP TCP/IP 栈的一部分。
*
*/
```

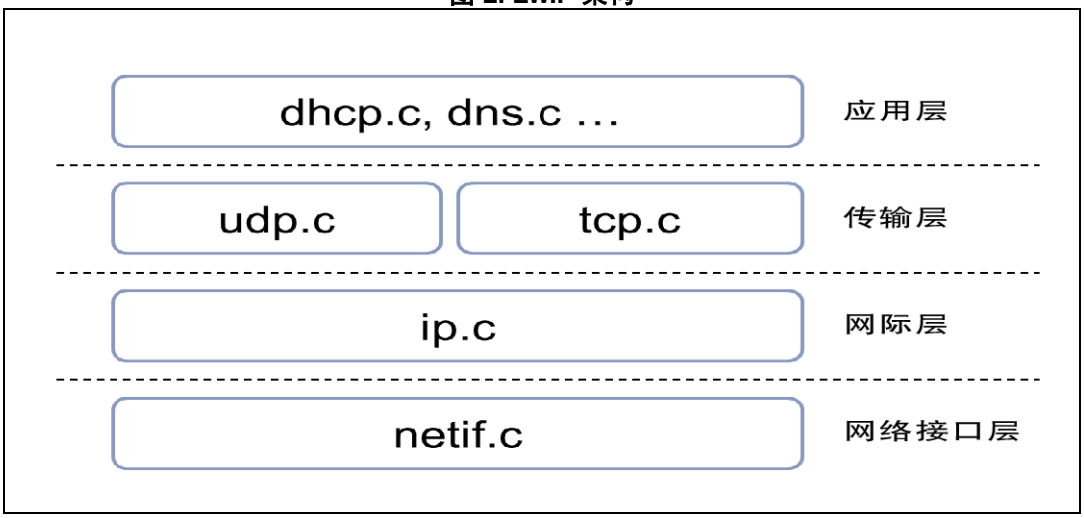
2.3 LwIP 架构

LwIP 符合 TCP/IP 模型架构，规定了数据的格式、传输、路由和接收，以实现端到端的通信。

此模型包括四个抽象层，用于根据涉及的网络范围，对所有相关协议排序（参见*图 2*）。这几层从低到高依次为：

- **链路层**包含了局域网的单网段 （链路）通信技术。
- **网际层 （IP）**将独立的网络连接起来，建立互联。
- **传输层**处理主机端口到主机端口的通信。
- **应用层**在实现多个应用进程相互通信的同时，完成应用所需的服务 （例如：数据处理）。

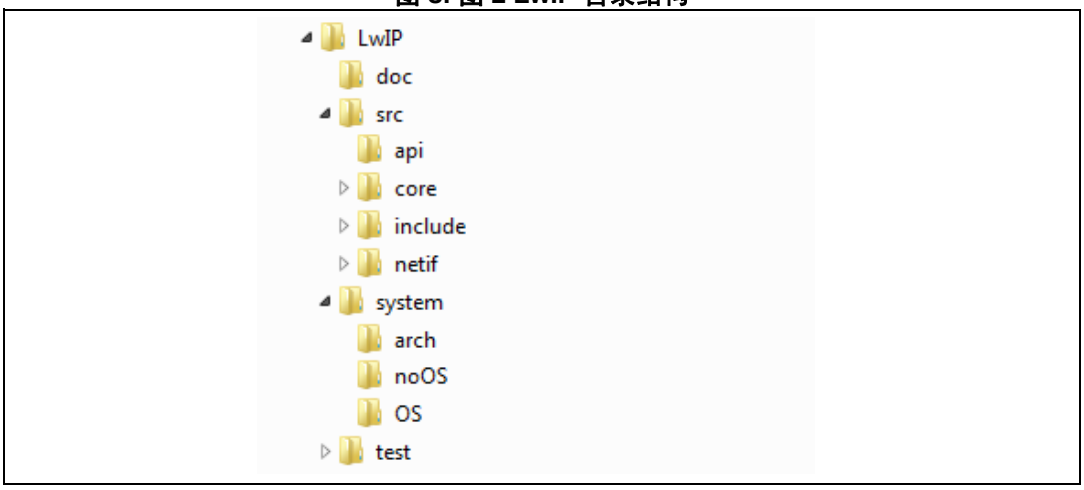
图 2. LwIP 架构



2.4 LwIP 栈的目录组织

解压之后，可在 \Middlewares\Third_Party\LwIP 下看到 LwIP 栈文件。

图 3. 图 2 LwIP 目录结构



其中

doc 包含文档文本文件
src 包含 LwIP 栈的源代码文件
api 包含 Netconn 和套接字 API 文件
core 包含 LwIP 内核文件
include 包含 LwIP 包含文件
netif 包含网络接口文件
system 包含 LwIP 端口硬件实现文件
arch 包含 STM32 架构端口文件 （所用的数据类型 ...）
OS 包含使用操作系统的 LwIP 端口硬件实现文件
noOS 包含独立模式的 LwIP 端口硬件实现文件

2.5 LwIP API 概述

如上所述， LwIP 栈提供了三种 API：

- Raw API
- Netconn API
- Socket API

2.5.1 Raw API

Raw API 基于原始 LwIP API。它可用于开发基于事件回调机制的应用。

当初始化应用时，用户需要为不同内核事件注册所需的回调函数（例如 TCP_Sent、TCP_error...）。当相应事件发生时， LwIP 会自发地调用相关的回调函数。

表 1 总结了 TCP 应用的 Raw API 函数。

表 1. TCP Raw API 函数

API 函数		说明
TCP 连接建立	tcp_new	建立一个新的 TCP PCB （协议控制块）。
	tcp_bind	将 TCP PCB 绑定到本地 IP 地址和端口。
	tcp_listen	启动 TCP PCB 上的监听进程。
	tcp_accept	注册回调函数，连接成功建立后调用。
	tcp_connect	发送连接建立请求。
发送 TCP 数据	tcp_write	将发送数据写入 TCP 缓冲区中。
	tcp_sent	注册回调函数，数据发送成功后调用。
	tcp_output	发送 TCP 缓冲区中的数据。
接收 TCP 数据	tcp_recv	注册回调函数， TCP 接收到数据后调用
应用轮询	tcp_poll	注册回调函数， TCP 慢定时器调用（500ms 一次）。
关闭并终止连接	tcp_close	主动关闭。
	tcp_err	注册回调函数，出错时调用。
	tcp_abort	中止连接，向远程主机发送 RST。

表 2 总结了 UDP 应用的 Raw API 函数。

表 2. UDP Raw API 函数

API 函数	说明
udp_new	创建新的 UDP PCB。
udp_remove	移除 UDP PCB 并释放相关资源。
udp_bind	将 UDP PCB 与本地 IP 地址和端口绑定。
udp_connect	建立 UDP PCB 远程 IP 地址和端口。
udp_disconnect	移除 UDP PCB 远程 IP 和端口。
udp_send	发送 UDP 数据。
udp_recv	注册回调函数，当收到新数据报时即对其调用。

2.5.2 Netconn API

Netconn API 为高层有序 API，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 LwIP TCP/IP 栈实现专用线程，并 / 或为应用实现多个线程。

表 3 总结了 Netconn API 函数。

表 3. Netconn API 函数

API 函数	说明
netconn_new	创建一个新连接。
netconn_delete	删除一个已有连接。
netconn_bind	将连接绑定到本地 IP 地址和端口。
netconn_connect	连接远程 IP 地址和端口。
netconn_send	通过 UDP 发送数据。
netconn_recv	接收数据。
netconn_listen	置 TCP netconn 处于监听模式。
netconn_accept	接受正在监听状态的 TCP 连接上的传入连接。
netconn_write	通过 TCP 数据 （将数据写入 TCP 缓冲区）。
netconn_close	主动关闭 TCP netconn。

2.5.3 Socket API

LwIP 提供了标准 BSD 套接字 API。它是有序 API，在内部构建于 Netconn API 之上。

表 4 总结了主要 socket API 函数。

表 4. Socket API 函数

API 函数	说明
socket	创建一个新套接字。
bind	将套接字绑定到 IP 地址和端口。
listen	监听套接字连接。
connect	将套接字连接到远程主机 IP 地址和端口。
accept	在套接字上接受新连接。
read	从套接字读取数据。
write	向套接字写入数据。
close	关闭套接字 （删除套接字）。

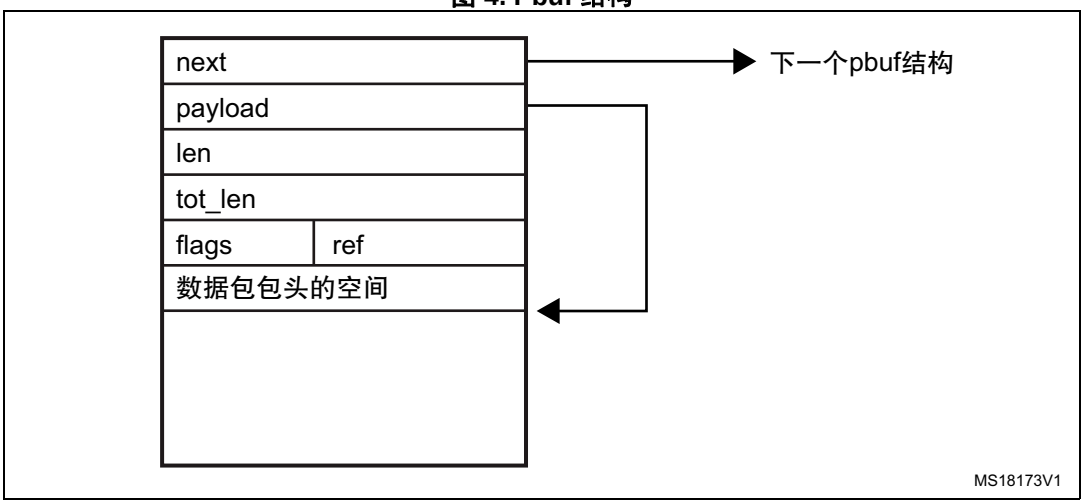
2.6 LwIP 缓冲管理

2.6.1 包缓冲结构

LwIP 使用名为 pbuf 的数据结构管理包缓冲。pbuf 结构可以通过动态内存申请 / 释放，并让包保留在静态内存中。

pbuf 为链表结构，因此数据包可以由多个 pbuf 组成 （链表）。

图 4. Pbuf 结构



其中

next 包含了指向 pbuf 链中下一个 pbuf 的指针

payload 包含了指向包数据载荷的指针

len 为 pbuf 数据内容长度

tot_len 为 pbuf 长度与链中后面 pbuf 的所有 len 字段之和

ref 为 4 位参考数，表示指向 pbuf 的指针数。只有 pbuf 的参考数为 0 时，才能将其从内存中释放。

flags （4 位）表示 pbuf 的类型。

LwIP 根据分配类型，定义了三种 pbuf：

- **PBUF_POOL**

  pbuf 动态分配 （内存池算法）。
- **PBUF_RAM**

  pbuf 动态分配 （内存堆算法）。
- **PBUF_ROM**

  不需为用户载荷分配内存空间：pbuf 载荷指针指向 ROM 内存中的数据，仅能用于发送常量数据。

对于包的接收，适合的 pbuf 类型为 PBUF_POOL，它允许从 pbuf 池中为收到的包快速分配内存。取决于所收包的大小，会分配一个或多个链接的 pbuf。PBUF_RAM 不适合包接收，因为此分配算法会造成延时。也可能导致内存碎片。

对于包的发送，用户可根据要发送的数据选择最适合的 pbuf 类型。

2.6.2 pbuf 管理 API

LwIP 有专门的 API 可与 pbuf 共同使用。该 API 实现于 pbuf.c 内核文件中。

表 5. Pbuf API 函数

API 函数	说明
pbuf_alloc	分配新的 pbuf。
pbuf_realloc	改变 pbuf 大小 （只能缩小）。
pbuf_ref	增加 pbuf 的参考数字段。
pbuf_free	释放 pbuf。
pbuf_clen	返回 pbuf 链中的 pbuf 数目。
pbuf_cat	将两个 pbuf 链接在一起 （但不会更改末尾 pbuf 链的参考数）。
pbuf_chain	将两个 pbuf 链接在一起 （增加尾链的参考数）。

表 5. Pbuf API 函数 （续）

API 函数	说明
pbuf_dechain	将第一个 pbuf 与链中后续的 pbuf 断开链接。
pbuf_header	调整载荷指针，隐藏或显示载荷中的头。
pbuf_copy_partial	将 （部分）包缓冲内容复制到应用提供的缓冲。
pbuf_take	将应用提供的数据复制到 pbuf 中。
pbuf_coalesce	从一个 pbuf 队列中创建单个 pbuf。
pbuf_memcmp	将指定偏移处的 pbuf 内容与其它内存比较。
pbuf_memfind	从某偏移开始，在 pbuf 中查找某内存。
pbuf_strstr	从某偏移开始，在 pbuf 中查找某字符串。

注： *“pbuf” 可为单个 pbuf 或 pbuf 链。*

当使用 Netconn API 时，则使用 netbuf （网络缓冲）发送 / 接收数据。

netbuf 只是 pbuf 结构的封装。它可容纳分配的或引用的数据。

提供了专用 API （在文件 netbuf.c 中实现）以管理 netbuf （分配、释放、链接、解压数据 ...）。

3 LwIP 与 STM32Cube 以太网 HAL 驱动之间的接口

此软件包包含两个实现：

- 无操作系统时的实现 （独立）
- 有操作系统时使用 CMSIS-RTOS API 的实现

对于两种实现，将 LwIP 移植至 STM32F4xx/STM32F2x7xx 的接口文件都位于 “lwip/system” 目录下。具体的驱动程序通过 ethernetif.c 文件来实现。

HAL 的以太网处理句柄 （ETH_HandleTypeDef）以及以太网 DMA 描述符（ETH_DMADescTypeDef）和以太网驱动的 Rx/Tx 缓冲区应在 *ethernetif.c* 文件中声明。

表 6 提供了 LwIP 接口 API 的说明。

表 6. 以太网接口函数说明

函数	说明
low_level_init	调用以太网驱动函数，初始化 STM32F4xx 和 STM32F2x7xx 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_init	初始化网络接口结构 （netif）并调用 low_level_init 以初始化以太网外设
ethernet_input	调用 low_level_input 接收包，然后将其提供给 LwIP 栈

下面的例子展示了怎样初始化以太网外设，它使用 HAL API，进入接口 API:

```
static void low_level_init(struct netif *netif)
{
  uint8_t macaddress[6]= {MAC_ADDR0, MAC_ADDR1, MAC_ADDR2, MAC_ADDR3,
MAC_ADDR4,    MAC_ADDR5};

  EthHandle.Instance = ETH;
  EthHandle.Init.MACAddr = macaddress;
  EthHandle.Init.AutoNegotiation = ETH_AUTONEGOTIATION_ENABLE;
  EthHandle.Init.Speed = ETH_SPEED_100M;
  EthHandle.Init.DuplexMode = ETH_MODE_FULLDUPLEX;
  EthHandle.Init.MediaInterface = ETH_MEDIA_INTERFACE_MII;
  EthHandle.Init.RxMode = ETH_RXINTERRUPT_MODE;
  EthHandle.Init.ChecksumMode = ETH_CHECKSUM_BY_HARDWARE;
  EthHandle.Init.PhyAddress = DP83848_PHY_ADDRESS;
```

```
  /* 配置以太网外设 （GPIO、时钟、 MAC、 DMA） */
  HAL_ETH_Init(&EthHandle) ;

  /* 初始化 Tx 描述符列表：链接模式 */
  HAL_ETH_DMATxDescListInit(&EthHandle, DMATxDscrTab, &Tx_Buff[0][0],
ETH_TXBUFNB);

  /* 初始化 Rx 描述符列表：链接模式 */
  HAL_ETH_DMARxDescListInit(&EthHandle, DMARxDscrTab, &Rx_Buff[0][0],
ETH_RXBUFNB);

  ...

/* 使能 MAC 和 DMA 发送和接收 */
  HAL_ETH_Start(&EthHandle);
}
```

ethernet_input() 函数的实现在独立模式和 RTOS 模式时是不同的：

- 在独立应用中，此函数必须被插入到应用的主循环中，以便轮询任何收到的包。
- 在 RTOS 应用中，此函数为一个阻塞线程，当得到所等待的信号量时才处理收到的数据包。当以太网外设收到数据并生成中断时，给出此信号量。

ethernetif.c 文件还为底层初始化（GPIO、CLK …）实现了以太网外设 MSP（MCU Support Package）程序和中断回调函数。

对于 RTOS 实现，还需使用其它文件（*sys_arch.c*）。此文件为 RTOS 服务实现了仿真层（共享内存的访问，信号量，邮箱）。此文件应根据所使用的 RTOS 调整，对于本软件包来说为 FreeRTOS。

4 LwIP 配置

LwIP 提供了名为 *lwipopts.h* 的文件，它允许用户充分配置栈及其所有模块。用户不需要定义所有 LwIP 选项：如果未定义某选项，则使用 *opt.h* 文件中定义的默认值。因此， *lwipopts.h* 提供了覆盖许多 lwIP 行为的方法。

4.1 模块支持

用户可为其应用选择他所需的模块，通过仅编译选定的特性优化了代码长度。

例如，若需要禁用 UDP 或者启用 DHCP （基于 UDP 实现），在 *lwipopts.h* 文件中分别需进行以下定义：

```
/* 禁用 UDP */
#define LWIP_UDP 0

/* 启用 DHCP */
#define LWIP_DHCP 1
```

4.2 内存配置

LwIP 提供了一种灵活的方法管理内存池的大小和组织。

它在数据段中保留了一个固定大小的静态内存区。它细分为不同的池，而 lwIP 将其用于不同的数据结构。例如，有一个 tcp_pcb 结构体的池，还有一个 udp_pcb 结构体的池。每个池都可配置为容纳固定数目的数据结构。该数目可在 *lwipopts.h* 文件中更改。例如，MEMP_NUM_TCP_PCB 和 MEMP_NUM_UDP_PCB 定义了在某一时间系统中可激活的 tcp_pcb 和 udb_pcb 结构的最大数目。

用户选项可在 *lwipopts.h* 中更改。*表 7* 总结了主要 RAM 内存选项。

表 7. LwIP 内存配置

LwIP 内存选项	定义
MEM_SIZE	LwIP 堆内存大小：用于所有 LwIP 动态内存分配。
MEMP_NUM_PBUF	MEM_REF 和 MEM_ROM pbuf 总数。
MEMP_NUM_UDP_PCB	UDP PCB 结构体的总数。
MEMP_NUM_TCP_PCB	TCP PCB 结构体的总数。
MEMP_NUM_TCP_PCB_LISTEN	处于监听状态的 TCP PCB 总数。
MEMP_NUM_TCP_SEG	同时在队列中的 TCP 段的最大数目。
PBUF_POOL_SIZE	PBUF_POOL 类型的 pbuf 总数。

表 7. LwIP 内存配置 （续）

LwIP 内存选项	定义
PBUF_POOL_BUFSIZE	PBUF_POOL 类型 pbuf 的大小。
TCP_MSS	TCP 最大段的大小。
TCP_SND_BUF	对于一个连接， TCP 的发送缓冲空间。
TCP_SND_QUEUELEN	TCP 发送队列中 pbuf 的最大数。
TCP_WND	声明的 TCP 接收窗大小。

5 使用 LwIP 栈开发应用

5.1 使用 Raw API 在独立模式中开发

5.1.1 工作模型

在独立模式中，工作模型基于轮询模式不停地检查是否收到了数据包。

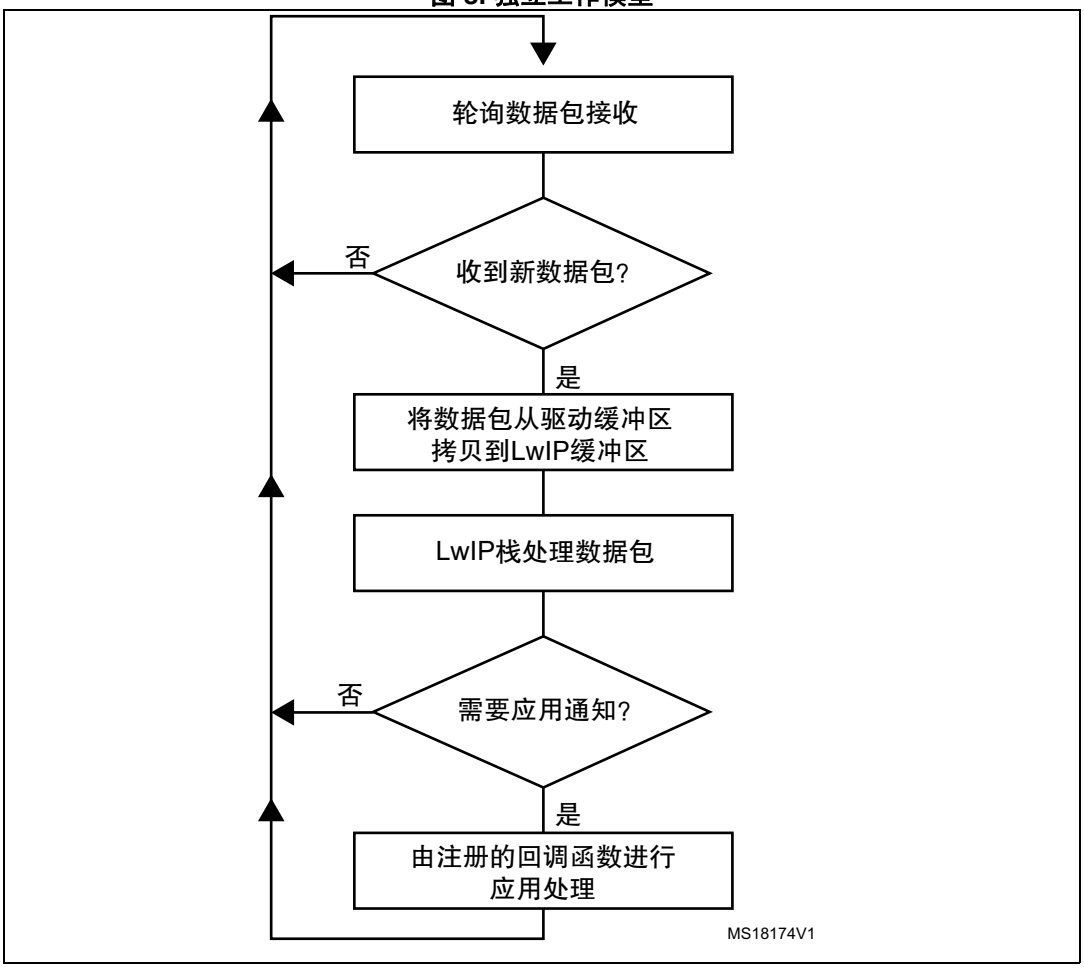
当收到包时，首先要将数据包从以太网接收缓冲区中拷贝到 LwIP 的协议栈缓冲区。为了更快的完成数据包的拷贝，应该从缓冲池 （PBUF_POOL）分配 LwIP 缓冲 （pbuf）。

拷贝完成后， LwIP 会对数据包进行处理。栈根据所收到的包确定是否通知应用层。

LwIP 使用事件回调机制与应用层通信。因此，应在进行通信之前，为相关事件注册回调函数。

请参考*图 5* 中的独立工作模型流图说明。

图 5. 独立工作模型



对于 TCP 应用，必须注册以下回调函数：

- TCP 连接建立时触发，通过 TCP_accept API 注册
- 接收到 TCP 数据包时触发，通过 TCP_recev API 注册
- 数据成功发送后触发，通过 TCP_sent API 注册
- TCP 出错时触发 （在 TCP 中止事件之后），通过 TCP_err API 注册
- 周期性触发 （1s 2 次），用于轮询应用，通过 TCP_poll API 注册

5.1.2 TCP 回响服务器演示举例

TCP 回响服务器示例在目录 *\LwIP\LwIP_TCP_Echo_Server* 中，它是一个 TCP 服务器的简单应用，可对从远程客户端收到的任何 TCP 数据包做出回响。

下面的例子提供了固件结构的说明。以下内容节选自 *main.c* 文件。

```
int main(void)
{
  /* 复位所有外设，初始化 Flash 接口和 Systick。 */
  HAL_Init();

  ...

  /* 初始化 LwIP 栈 */
   lwIP_init();
   /* 网络接口配置 */
   Netif_Config();
  ...

/* tcp 回响服务器初始化 */
  tcp_echoserver_init();

  /* 无限循环 */
  while (1)
  {
    /* 从以太网缓冲区中读取数据包，交给
       LwIP 处理 */
    ethernetif_input(&gnetif);

    /* 处理 LwIP 超时 */
    sys_check_timeouts();
  }
}
```

其中调用了下列函数：

1. *HAL_Init* 函数调用的目的是复位所有外设，并初始化 Flash 接口和 Systick 定时器
2. *lwIP_init* 函数调用的目的是初始化 LwIP 栈内部结构体，并开始栈操作。
3. *Netif_config* 函数调用的目的是配置网络接口 （netif）。
4. *tcp_echoserver_init* 函数调用的目的是初始化 TCP 回响服务器应用。
5. 在无限 while 循环中的 *ethernetif_input* 函数轮询包的接收。当收到包时，将包传给栈处理
6. *sys_check_timeouts* LwIP 函数调用的目的是处理某些 LwIP 内部周期性任务 （协议定时器、 TCP 包的重传 ...）。

tcp_echoserver_init 函数描述

tcp_echoserver_init 函数代码如下：

```
void tcp_echoserver_init(void)
{
  /* 创建新的 tcp pcb */
  tcp_echoserver_pcb = tcp_new();

  if (tcp_echoserver_pcb != NULL)
  {
    err_t err;

    /* 将 echo_pcb 绑定到端口 7 （ECHO 协议） */
    err = tcp_bind(tcp_echoserver_pcb, IP_ADDR_ANY, 7);

    if (err == ERR_OK)
    {
      /* echo_pcb 开始 tcp 监听 */
      tcp_echoserver_pcb = tcp_listen(tcp_echoserver_pcb);

      /* 注册 LwIP tcp_accept 回调函数 */
      tcp_accept(tcp_echoserver_pcb, tcp_echoserver_accept);
    }
    else
    {
      /* 释放 pcb */
      memp_free(MEMP_TCP_PCB, tcp_echoserver_pcb);
    }
  }
}
```

LwIP API 调用 *tcp_new* 来分配一个新的 TCP 协议控制块（PCB）（tcp_echoserver_pcb）。

使用 *tcp_bind* 函数，将分配的 TCP PCB 绑定到本地 IP 地址和端口。

绑定 TCP PCB 之后，会调用 *tcp_listen* 函数以在 TCP PCB 上开始 TCP 监听进程。

最后，应给 *tcp_echoserver_accept* 回调函数赋值，以处理 TCP PCB 上传入的 TCP 连接，这通过使用 *tcp_accept* LwIP API 函数完成。

从这点开始， TCP 服务器已经准备好接收任何来自远程客户端的连接。

tcp_echoserver_accept 函数描述

下面的例子展示了怎样使用 tcp_echoserver_accept 用户回调函数，处理传入的 TCP 连接。以下内容节选自该函数。

```
static err_t tcp_echoserver_accept(void *arg, struct tcp_pcb *newpcb, err_t
err)
{
  ...
/* 分配结构体 es 以保存 tcp 连接信息 */
  es = (struct tcp_echoserver_struct *)mem_malloc(sizeof(struct
tcp_echoserver_struct));
  if (es != NULL)
  {
    es->state = ES_ACCEPTED;
    es->pcb = newpcb;
    es->p = NULL;

    /* 将新分配的 es 结构体作为参数传给 newpcb */
    tcp_arg(newpcb, es);

    /* 为 newpcb 注册 lwIP tcp_recv 回调函数 */
    tcp_recv(newpcb, tcp_echoserver_recv);

    /* 为 newpcb 注册 lwIP tcp_err 回调函数 */
    tcp_err(newpcb, tcp_echoserver_error);

    /* 为 newpcb 注册 lwIP tcp_poll 回调函数 */
    tcp_poll(newpcb, tcp_echoserver_poll, 1);

    ret_err = ERR_OK;
  ...
}
```

其中调用了下列函数：

1. 通过 newpcb 参数，将新的 TCP 连接传给 *tcp_echoserver_accept* 回调函数。
2. es 结构体被用来存储应用状态。通过调用 *tcp_arg LwIP API*，将它作为一个参数传给 TCP PCB “newpcb” 连接。
3. 通过调用 *LwIP API tcp_recv*，为 TCP 接收回调函数 *tcp_echoserver_recv* 赋值。此回调处理远程客户端的所有数据流。
4. 通过调用 *LwIP API tcp_err*，为 TCP 错误回调函数 *tcp_echoserver_error* 赋值。此回调处理 TCP 错误。
5. 通过调用 *LwIP API tcp_poll*，为 TCP 轮询回调函数 *tcp_echoserver_poll* 赋值，以处理周期性的应用任务 （例如检查是否还有应用数据要发送）。

5.2 使用 Netconn 或 Socket API 基于 RTOS 开发

5.2.1 工作模型

使用 RTOS 的工作模型有如下特点：

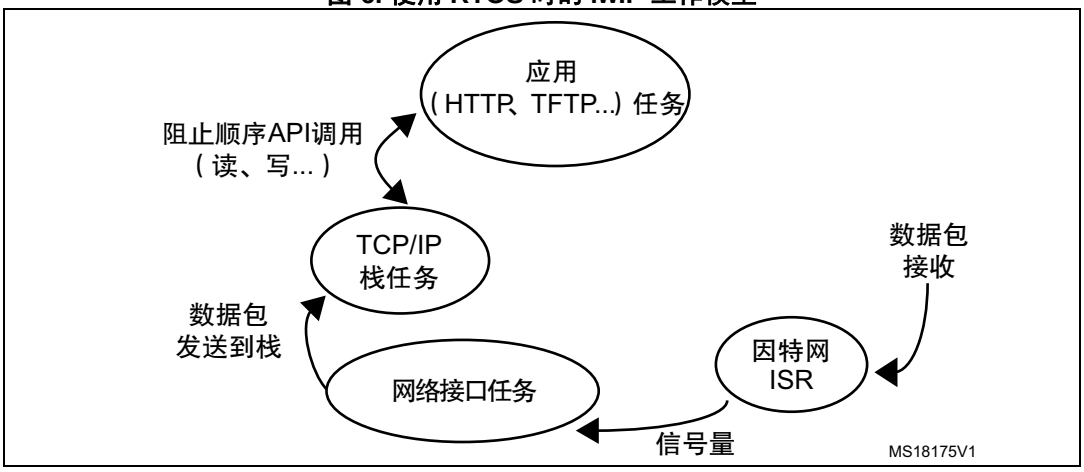
TCP/IP 栈和应用运行在不同的线程中。

应用通过有序 API 调用与栈通信，它使用 RTOS 邮箱机制进行进程间通信。 API 调用为阻塞调用。这意味着在从栈收到响应之前，应用线程阻塞。

使用另外一个线程 —— 网络接口线程 —— 用于将驱动缓冲区收到的数据包拷贝至 LwIP 协议栈缓冲区。此进程由以太网接收中断所释放的信号量唤醒。

请参考*图 6* 中使用 RTOS 时的 lwIP 工作模型流程图说明。

图 6. 使用 RTOS 时的 lwIP 工作模型



5.2.2 使用 Netconn API 的 TCP 回响服务器演示举例

从应用的角度来看，Netconn API 提供了一种比 raw API 更简单的方法来开发 TCP/IP 应用。这是因为它有一个更加直观的有序 API。

下面的例子显示了使用 Netconn API 开发的 TCP 回响服务器应用。以下内容节选自 *main.c* 文件。

```
int main(void)
{
  ...
  /* 创建并开始线程 */
  osThreadDef(Start, StartThread, osPriorityNormal, 0,
configMINIMAL_STACK_SIZE * 2);
  osThreadCreate (osThread(Start), NULL);

  /* 开始调度器 */
  osKernelStart (NULL, NULL);

  /* 程序不应该运行到这里，因为现在调度器在控制 */
  for( ;; );
}
```

开始线程有如下代码：

```
static void StartThread(void const * argument)
{
  ...
   /* 创建 tcp_ip 栈线程 */
   tcpip_init( NULL, NULL );

   /* 网络接口配置 */
   Netif_Config();

   /* 初始化 tcp 回响服务器 */
  tcpecho_init();

  for( ;; )
  {
  }
}
```

其中调用了下列函数：

1. 调用了 tcpip_init 函数，对 LwIP 栈模块初始化并开始 TCP/IP 栈线程。
2. Netif_config 函数调用的目的是配置网络接口 （netif）。
3. TCP 回响服务器线程在 *tcpecho_init* 函数中创建。

```
void tcpecho_init(void)
{
  sys_thread_new("tcpecho_thread", tcpecho_thread, NULL,
DEFAULT_THREAD_STACKSIZE,     TCPECHO_THREAD_PRIO);
}
```

tcpecho_thread 函数说明

TCP 回响服务器线程有如下代码：

```
static void tcpecho_thread(void *arg)
{
  /* 创建一个新连接标识符。 */
  conn = netconn_new(NETCONN_TCP);

  if (conn!=NULL)
  {
    /* 将连接绑定至已知的端口号 7。 */
    err = netconn_bind(conn, NULL, 7);

    if (err == ERR_OK)
    {
      /* 告知连接进入监听模式。 */
      netconn_listen(conn);

      while (1)
      {
        /* 抓取新连接。 */
        accept_err = netconn_accept(conn, &newconn);

        /* 处理新连接。 */
        if (accept_err == ERR_OK)
        {
          while (( recv_err = netconn_recv(newconn, &buf)) == ERR_OK)
          {
            do
            {
              netbuf_data(buf, &data, &len);
              netconn_write(newconn, data, len, NETCONN_COPY);
            }
            while (netbuf_next(buf) >= 0);

            netbuf_delete(buf);
          }
          /* 关闭连接，丢弃连接标识符。 */
          netconn_close(newconn);
          netconn_delete(newconn);
```

```
        }
      }
    }
    else
    {
      netconn_delete(newconn);
    }
  }
}
```

其中执行了下述序列：

1. 调用了 *Netconn_new* API 函数，参数 NETCONN_TCP 将创建一个新 TCP 连接。
2. 之后，将新创建的连接绑定到端口 7 （回响协议），方法是调用 *Netconn_bind* API 函数。
3. 绑定连接之后，通过调用 *Netconn_listen* API 函数，应用开始监听连接。
4. 在无限 while(1) 循环中，通过调用 API 函数 *Netconn_accept*，应用等待一个新连接。当没有传入的连接时，进程被阻塞。
5. 当有传入的连接时，通过调用 *netconn_recv* API 函数，应用可开始接收数据。传入的数据接收在 netbuf 中。
6. 应用可通过调用 *netbuf_data netbuf* API 函数得到接收的数据。
7. 通过调用 *Netconn_write* API 函数，将接收的数据发送回 （回响）远程 TCP 客户端。
8. *Netconn_close* 和 *Netconn_delete* 分别用于关闭和删除 Netconn 连接。

6 LwIP 包描述

6.1 LwIP 包目录

该包包含了一组应用，运行于 LwIP 栈、 STM32Cube HAL 和 BSP 驱动之上。固件包含下列模块：

- **驱动**：包含 STM32F4xx/STM32F2x7xx 微控制器的底层驱动。
  - CMSIS
  - BSP 驱动器
  - HAL 驱动器
- **中间件**：包含库和协议组件
  - LwIP TCP/IP 栈
  - FatFS
  - FreeRTOS
- **项目**：包含源文件和每个应用的配置

应用位于项目库的此路径之下：*Projects****STM322xx_EVAL***LwIP*\ *和* *Projects****STM324xx_EVAL***LwIP*\。

6.2 应用设置

6.2.1 PHY 接口配置

以太网外设与外部 PHY 接口，提供物理层通信。 PHY 寄存器定义和定义语句位于 HAL 配置文件 *stm32f4xx_hal_conf.h* 中。

PHY 可工作于 MII 或 RMII 模式。若要选择所需的模式，请在初始化以太网外设时在 Init 结构体中填入 MediaInterface 参数。

注： *当使用 STM324x9I-EVAL 板时，不支持 RMII 模式。*

当使用 STM324xG-EVAL 板工作于 RMII 模式时，用户必须提供 50 MHz 时钟，方法是在 CN3 下的 U3 尺寸上，从 JP5 移除跳线，焊接一个 50 MHz 振荡器 （参考 SM7745HEV-50.0M 或同类器件）。该振荡器不随板提供。

6.2.2 MAC 和 IP 地址设置

默认 MAC 地址设为 00:00:00:00:00:02。若要更改此地址，请修改 *stm32f4xx_hal_conf.h* 文件中定义的六个字节。

默认 IP 地址设为：192.168.0.10。若要修改此地址，请修改 *main.h* 文件中定义的四个字节。

6.2.3 固件特性

此软件还具备一些扩展模块。

支持 DHCP 协议，因此当 STM32 MCU 连接到 DHCP 服务器时，它可作为 DHCP 客户端得到动态 IP 地址。 若要启用 DHCP 协议，请取消下面宏定义的注释：

`#define USE_DHCP" from main.h file.`

注： *如果配置为通过 DHCP 获取 IP 地址，但应用无法在它已经连接到的网络上发现 DHCP 服务器，则 IP 地址会自动设为静态地址 （192.168.0.10）。*

用户可通过在 *main.h* 中定义宏 `#define USE_LCD` ，启用 LCD 控制器。如果启用，将显示文本消息告知用户应用的状态 （分配的 IP 地址、网络链路状态 ...）

注： *入门级的应用不支持 DHCP 或 LCD 模块。更多信息，请参见第 7 章节：使用 LwIP 应用。*

6.3 评估板设置

6.3.1 STM324x9I-EVAL 设置

若需在 STM324x9I-EVAL 板上运行软件，请如*表 8* 中所示配置。

表 8. STM324x9I-EVAL 跳线配置

跳线	MII 模式配置
JP6	1-2：由外部晶振提供 25 MHz 时钟 2-3：由 PA8 处的 MCO 提供 25 MHz 时钟

6.3.2 STM324xG-EVAL 设置

若需在 STM324xG-EVAL 板上运行软件，请如*表 9* 中所示配置。

表 9. STM324xG-EVAL 跳线配置

跳线	MII 模式配置	RMII 模式配置
JP5	1-2：由外部晶振提供 25 MHz 时钟 2-3：由 PA8 处的 MCO 提供 25 MHz 时钟	不适用
JP6	2-3：启用 MII 接口模式。	1-2：启用 RMII 接口模式。
JP8	开启：MII 接口模式被选中。	关闭：RMII 接口模式被选中。

6.3.3 STM3222xG-EVAL 设置

若需在 STM322xG-EVAL 板上运行软件，请如*表 10* 中所示配置。

表 10. STM322xG-EVAL 跳线配置

跳线	MII 模式配置	RMII 模式配置
JP5	1-2：由外部晶振提供 25 MHz 时钟 2-3：由 PA8 处的 MCO 提供 25 MHz 时钟	不适用
JP6	2-3：启用 MII 接口模式。	1-2：启用 RMII 接口模式。
JP8	开启：MII 接口模式被选中。	关闭：RMII 接口模式被选中。

7 使用 LwIP 应用

STM32Cube LwIP 包带有多个应用，使用了不同 LwIP 栈的 API 集。

如*表 11* 中所示，这些应用分为三类。

表 11. LwIP 应用分类

分类	应用
入门级 （基础）	TCP 回响客户端
	TCP 回响服务器
	UDP 回响客户端
	UDP 回响服务器
	TCP 和 UDP 回响服务器 （Netconn API）
特性	HTTP 服务器 （Raw API）
	HTTP 服务器 （Netconn API）
	HTTP 服务器 （Socket API）
集成	TFTP 服务器

入门级应用使用最低的配置，在 LwIP 栈之上运行应用。使用 LED 通知用户应用状态。

特性级应用提供更多的灵活性和选项。它们支持 HTTP、 DHCP 等网络协议，使用 LCD 消息指示应用状态。

集成级应用支持 FatFS 中间件组件和 TFTP 协议，以在评估板上的 microSD™ 卡上收发文件。

7.1 入门级应用

7.1.1 TCP 回响客户端

此应用的作用是测试基本的 TCP 连接。 STM32 MCU 作为 TCP 客户端，连接到 TCP 服务器。客户端发送字符串，服务器将同样的字符串回响给客户端。

若需测试 TCP 回响客户端应用，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 生成演示代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
   LED 指示了 LwIP 初始化是成功还是失败 （此应用不支持动态地址分配 “DHCP”）。
3. 在远程 PC 上，打开命令行提示窗口。在 Windows 中，选择**开始 > 所有程序 > 附件 > 命令行提示**。

4. 在命令行提示，输入：

```
C:\>echotool /p tcp /s
```

其中：

– /p tcp 为 TCP 协议 （TCP 协议）

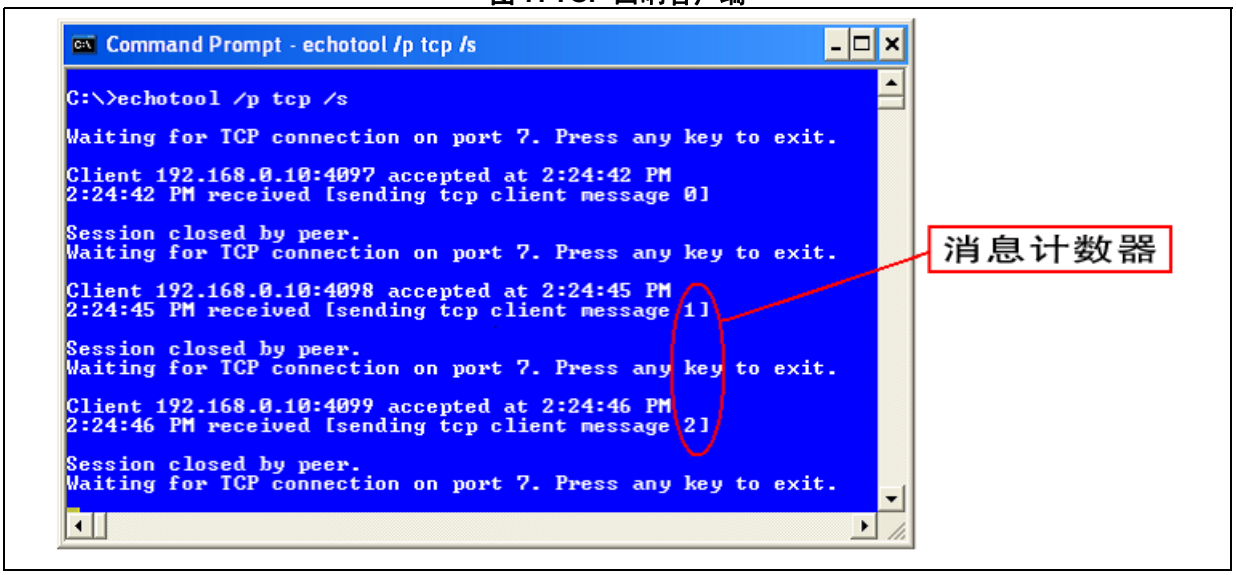
– /s 为连接的实际模式 （服务器模式）

5. 当按下 STM324xx-EVAL/STM322xG-EVAL 板上的 Key 按键时，客户端会发送字符串，服务器将相同的字符串回响给客户端。

注： *请确认远程 PC IP 地址与 main.h 文件中定义的相同 （默认为 192.168.0.11）。*

图 7 显示了此命令字符串和模块响应的例子。

图 7. TCP 回响客户端



7.1.2 TCP 回响服务器

此应用的作用是测试基本的 TCP 连接。 STM32 MCU 作为 TCP 服务器，等待客户端请求。它仅回响发出的内容。

若需测试 TCP 回响服务器演示程序，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 生成演示代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
   LED 指示了 LwIP 初始化是成功还是失败 （此应用不支持动态地址分配 “DHCP”）。
3. 在远程 PC 上，打开命令行提示窗口。对于 Windows，选择**开始 > 所有程序 > 附件 > 命令行提示**。

4. 在命令行提示，输入：

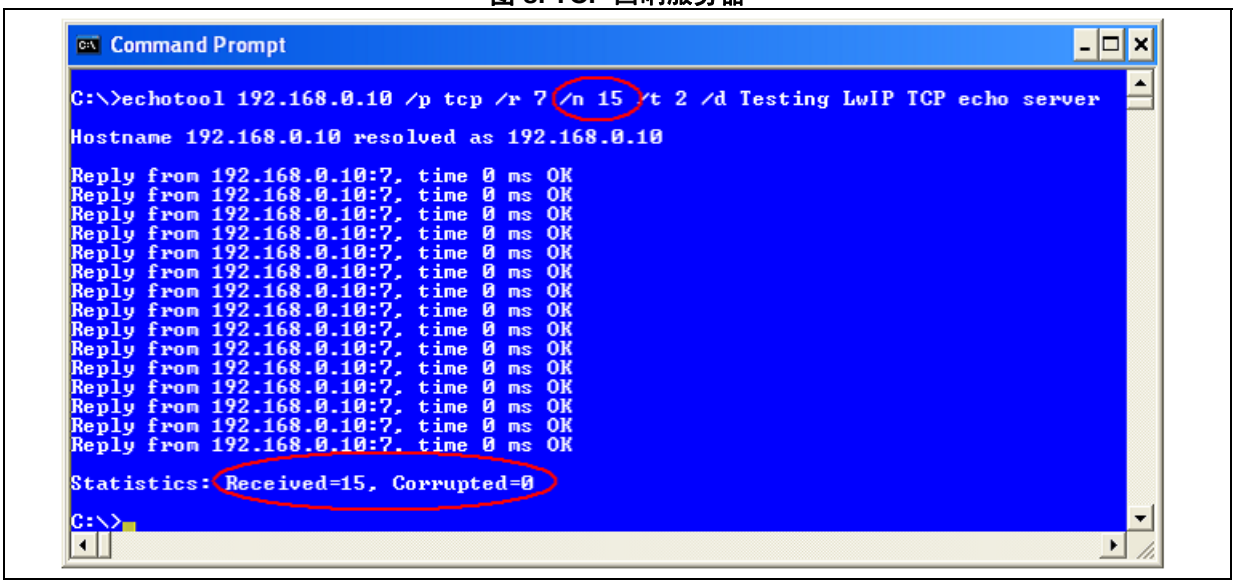
```
C:\>echotool IP_address /p tcp /r 7 /n 15 /t 2 /d Testing LwIP TCP echo
server
```

其中：

- `IP_address` 为实际板子的 IP 地址。默认情况下，会使用静态 IP 地址：192.168.0.10
- `/p` tcp 为协议 （TCP 协议）
- `/r` 为回响服务器的实际远程端口 （回响端口）
- `/n` 为回响请求的数目 （例如， 15）
- `/t` 为连接超时时间，单位为秒 （例如， 2）
- `/d` 为要为回响发送的消息 （例如， “Testing LwIP TCP echo server”）

图 8 显示了此命令字符串和模块响应的例子。

图 8. TCP 回响服务器



注： *提供的统计指标为测试结束时收到的和毁坏的包数。*

7.1.3 UDP 回响客户端

此应用的作用是测试基本的 UDP 回响连接。STM32 MCU 作为 UDP 客户端，连接到 UDP 服务器。

若需测试 UDP 回响客户端演示程序，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 生成演示代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
   LED 指示了 LwIP 初始化是成功还是失败 （此应用不支持动态地址分配 “DHCP”）。
3. 在远程 PC 上，打开命令行提示窗口。对于 Windows，选择**开始 > 所有程序 > 附件 > 命令行提示**。
4. 在命令行提示，输入：

```
C:\>echotool /p udp /s
```

其中：

– /p udp 为协议 （UDP 协议）

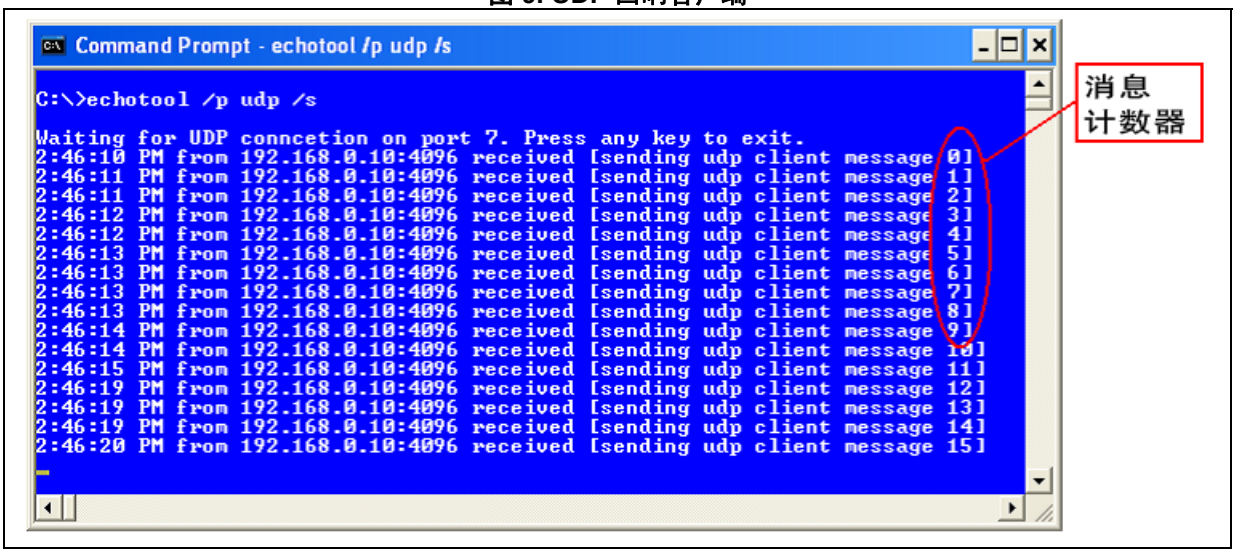
– /s 为连接的实际模式 （服务器模式）

5. 当按下 STM324xx-EVAL/STM322xG-EVAL 板上的 Key 按键时，客户端会发送字符串，服务器将相同的字符串回响给客户端。

注： *请确认远程 PC IP 地址与 main.h 文件中定义的相同 （默认为 192.168.0.11）。*

图 9 显示了此命令字符串和模块响应的例子。

图 9. UDP 回响客户端



7.1.4 UDP 回响服务器

此应用的作用是测试基本的 UDP 连接。 STM32 MCU 作为 UDP 服务器，等待客户端请求。

若需测试 UDP 回响服务器应用，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 生成演示代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
   LED 指示了 LwIP 初始化是成功还是失败 （此应用不支持动态地址分配 “DHCP”）。
3. 在远程 PC 上，打开命令行提示窗口。对于 Windows，选择 ***开始 > 所有程序 > 附件 > 命令行提示***。
4. 在命令行提示，输入：

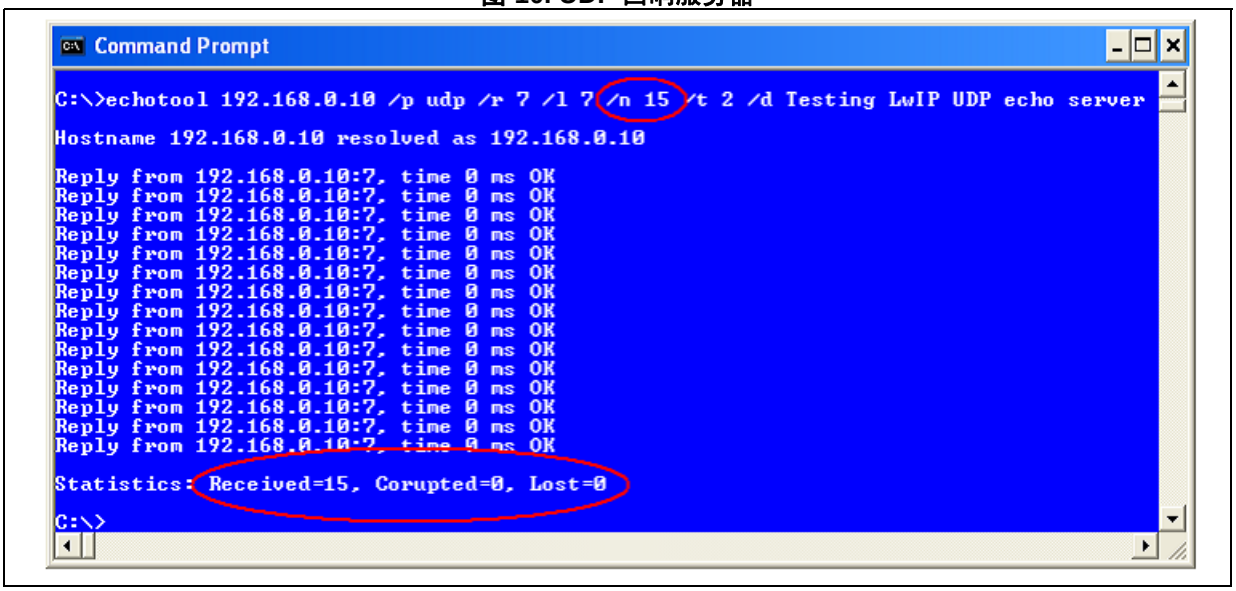
```
C:\>echotool IP_address /p udp /r 7 l/ 7 /n 15 /t 2 /d Testing
LwIP UDP echo server
```

其中：

- `IP_address` 为实际板子的 IP 地址。默认情况下，会使用静态 IP 地址：192.168.0.10
- `/p` 为协议 （UDP 协议）
- `/r` 为回响服务器的实际远程端口 （回响端口）
- `/l` 为客户端的实际本地端口 （回响端口）
- `/n` 为回响请求的数目 （例如， 15）
- `/t` 为连接超时时间，单位为秒 （例如， 2）
- `/d` 为要为回响发送的消息 （例如， “Testing LwIP UDP echo server”）

图 10 显示了此命令字符串和模块响应的例子。

图 10. UDP 回响服务器



注： *提供的统计指标为测试结束时收到的和毁坏的包数。*

7.1.5 基于 netconn API 的 UDP TCP 回响服务器

此演示程序提供了 TCP 和 UDP 协议的回响服务应用：

- 若需在TCP服务器模式中测试UDP TCP回响服务器netconn演示程序，请参考*第7.1.2章节：TCP 回响服务器*。
- 若需在UDP服务器模式中测试UDP TCP回响服务器netconn演示程序，请参考*第7.1.4章节：UDP 回响服务器*。

7.2 特性级应用

7.2.1 基于 raw API 的 Web 服务器

此应用实现了基于 LwIP raw API 的 Web 服务器。此应用中， STM32 MCU 作为 Web 服务器，可为远程 Web 浏览器提供 HTTP 服务。

该 Web 服务器应用实现了如下特性：

- URL 解析
- CGI （通用网关接口）
- SSI （服务器端嵌入）
- 动态头生成
- HTTP Post 请求

若需测试 Web 服务器应用，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 在 *main.h* 文件中，取消注释 “USE_DHCP” 或 “USE_LCD” 选项以启用 DHCP 客户端或 LCD 显示特性。
3. 生成应用代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
4. 若定义了 “USE_ DHCP” 及 “USE_LCD”，则 LCD 屏幕上会显示消息，指示 DHCP IP 地址分配是成功还是失败，否则 LED 会显示此操作的结果。
5. 当指定 IP 地址后 （动态或静态地址），用户可启动应用。
6. 在远程 PC 上，打开 Web 客户端 （Mozilla Firefox 或 Internet Explorer），并在 web 浏览器中键入板子的 IP 地址。默认情况下，会使用静态 IP 地址：192.168.0.10。

图 11. Web 服务器主页

服务器端嵌入 （SSI）

SSI 方法用于动态包含 HTML 代码中的动态数据。

这是通过在网页的 HTML 代码内放置特定的标签做到的。标签应为如下格式：

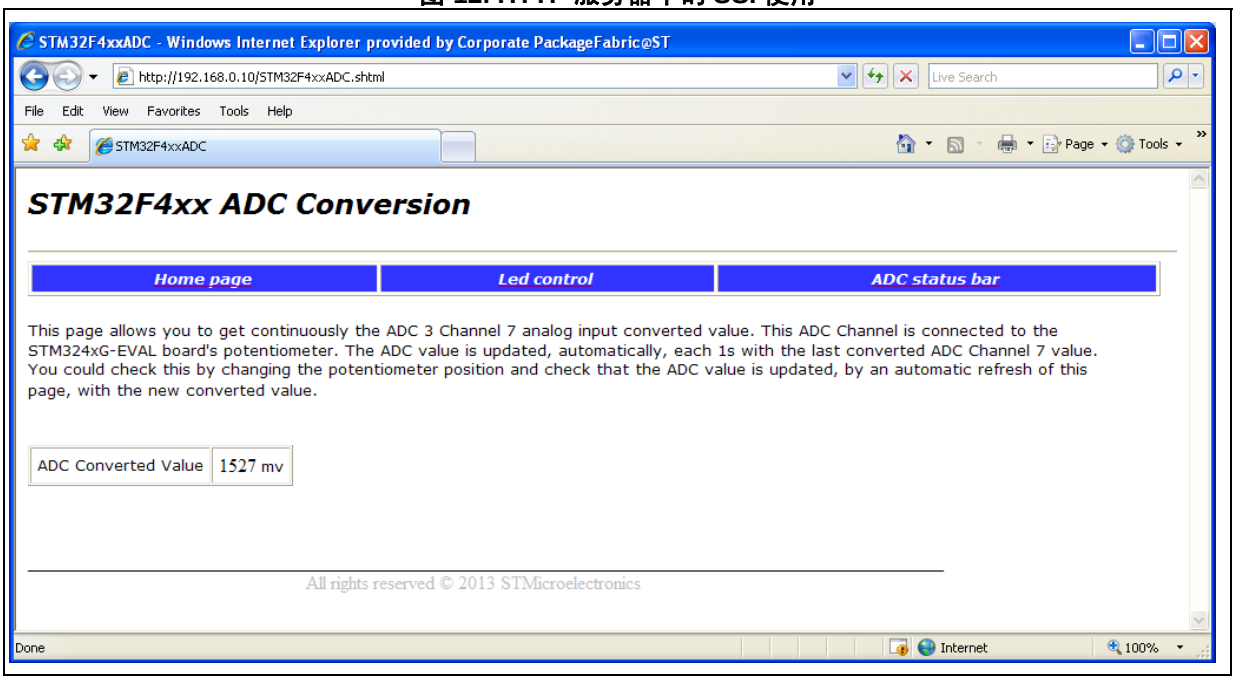
```
<!--#tag-->
```

对于 ADC 转换页面， HTML 代码内使用的是如下标签：

```
<!--#t-->
```

当有 ADC 网页 （它是 “.shtml” 扩展名）的请求时，服务器会解析网页，当发现该标签时，会将其替换为 ADC 转换值。

图 12. HTTP 服务器中的 SSI 使用



通用网关接口 （CGI）

CGI 为标准 web 技术，用于在服务器端执行来自客户端的请求，然后向客户端发送响应。

在 LwIP 中，提供的 CGI 仅支持 GET 方法请求，最多可处理编码在 URI 中的 16 个参数。服务器端执行的 CGI 处理程序会返回 HTTP 服务器发送给客户端的 HTML 文件。

在 HTTP 服务器演示程序中，此方法用于控制评估板的四个 LED（LED1、LED2、LED3 和 LED4）。

7.2.2 基于 netconn API 的 Web 服务器

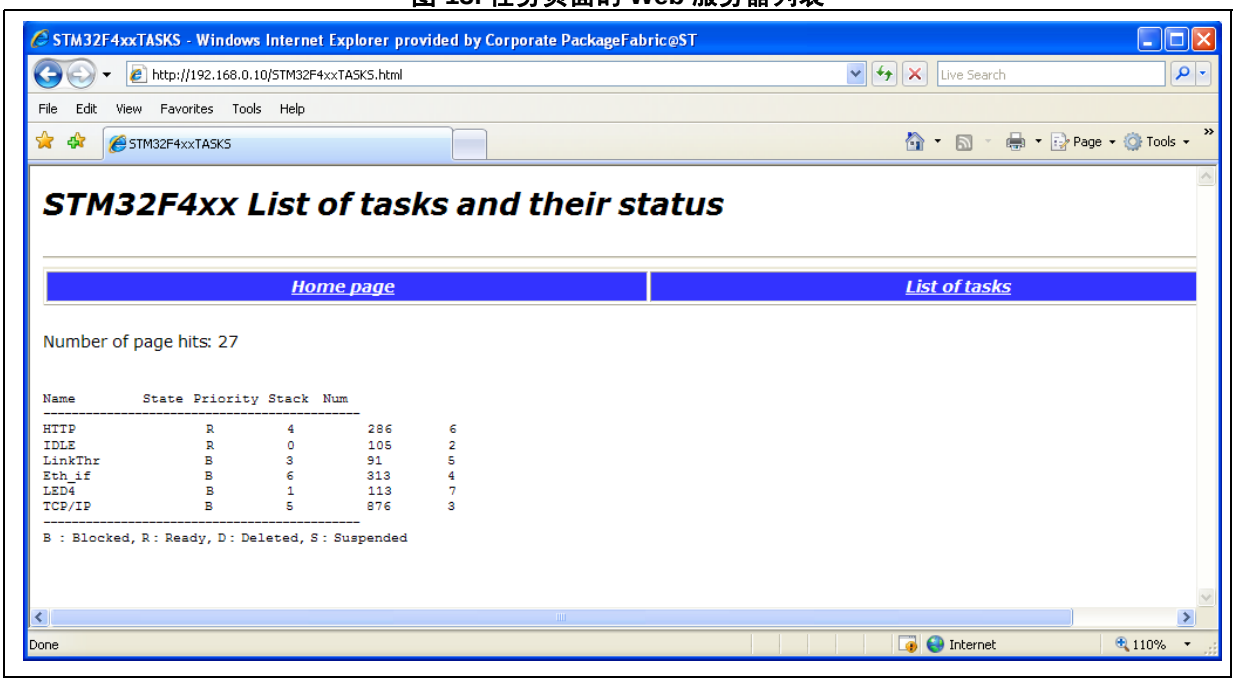
此应用实现了基于 netconn API 的 Web 服务器。此应用中，STM32 MCU 作为 Web 服务器，可为远程 Web 浏览器提供 HTTP 服务。

此 Web 服务器包含两个 HTML 页面。第一个页面给出了 STM32F4xx/STM32F2x7xx 微控制器和 LwIP 栈的一般信息。第二个页面列出了运行的任务和它们的状态。此页面每秒更新一次（参见*图 13*）。

若需测试 HTTP 服务器 netconn 演示程序，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 在 *main.h* 文件中，取消注释 “USE_DHCP” 或 “USE_LCD” 选项以启用 DHCP 客户端或 LCD 显示特性
3. 生成应用代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
4. 若定义了 “USE_ DHCP” 及 “USE_LCD”，则 LCD 屏幕上会显示消息，指示 DHCP IP 地址分配是成功还是失败，否则 LED 会显示此操作的结果。
5. 当指定 IP 地址后 （动态或静态地址），用户可启动应用。
6. 在远程 PC 上，打开 Web 客户端 （Mozilla Firefox 或 Internet Explorer），并在 web 浏览器中键入板子的 IP 地址。默认情况下，会使用静态 IP 地址：192.168.0.10。

图 13. 任务页面的 Web 服务器列表



7.2.3 基于 socket API 的 Web 服务器

此应用基于 socket API 实现了一个 Web 服务器。若需测试此演示程序，请参考*第 7.2.2 章节：基于 netconn API 的 Web 服务器*。

7.3 集成式应用

7.3.1 TFTP 服务器

TFTP 服务器提供文件传输服务，需要一个远程 TFTP 客户端。它可以从 STM324xx-EVAL/STM322xG-EVAL 板的 microSD 卡上收发文件。

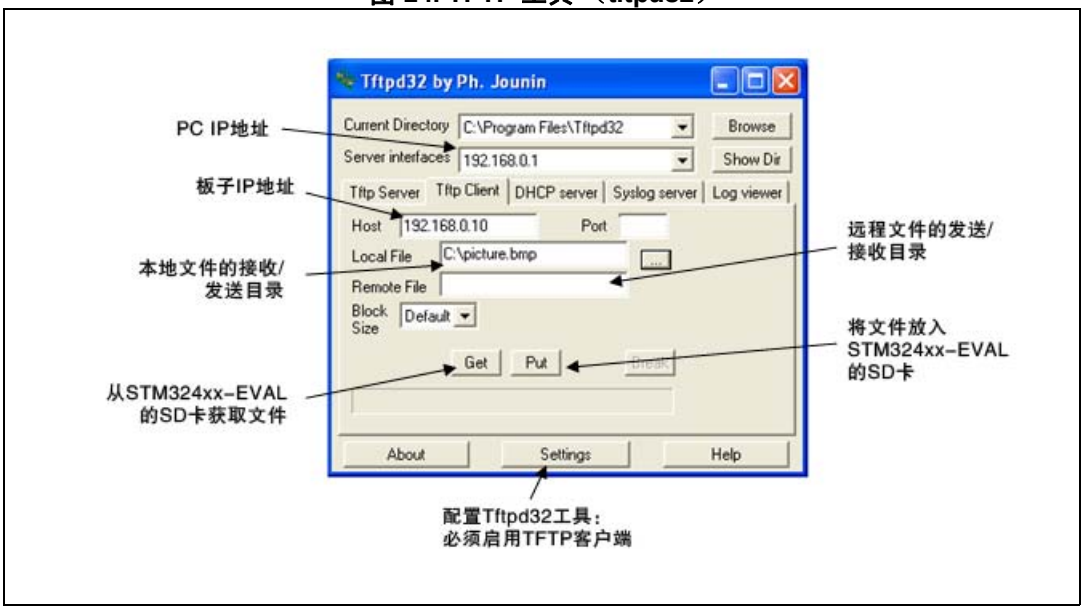
TFTP 服务器等待来自远程 TFTP 客户端的请求。必须通过远程 PC 连接 STM324xx-EVAL/STM322xG-EVAL 评估板，以上传或下载文件。为此，远程 PC 上必须安装 TFTP 客户端。这可通过使用 tftpd32 工具做到，它可以从 *http://tftpd32.jounin.net* 下载。

若需测试 TFTP 服务器应用，请遵循如下步骤：

1. 确认 STM324xx-EVAL/STM322xG-EVAL 跳线设置正确。
2. 在 *main.h* 文件中，取消注释 “USE_DHCP” 或 “USE_LCD” 选项以启用 DHCP 客户端或 LCD 显示特性。
3. 生成应用代码并编程到 STM32F4xx/STM32F2x7xx 闪存。
4. 若定义了 “USE_ DHCP” 及 “USE_LCD”，则 LCD 屏幕上会显示消息，指示 DHCP IP 地址分配是成功还是失败，否则 LED 会显示此操作的结果
5. 当指定 IP 地址后 （动态或静态地址），用户可启动应用。
6. 在远程 PC 上，打开 TFTP 客户端 （例如， TFTPD32）并配置 TFTP 服务器地址（TFTPD32 中的主机地址）。
7. 开始在 STM324xx-EVAL/STM322xG-EVAL 板的 micro SD 上收发文件。

图 11 给出了 tftpd32 工具概述。

图 14. TFTP 工具 （tftpd32）



注： *在 STM324xx-EVAL/STM322xG-EVAL 板上传 / 下载文件之前，请确认 microSD 卡已插入专用的连接器。*

8 结论

LwIP 包可通过 STM32Cube HAL API 使用 lwIP TCP/IP 栈。此开源栈在保持相对较低 RAM/ROM 使用的同时，提供了完整的 TCP/IP 栈服务。

开发 TCP/IP 应用有两种方法：使用独立模式，或者使用实时操作系统（RTOS）进行多线程工作。

附录 A FAQ

A.1 我怎样选择静态或动态 （DHCP）IP 地址分配？

当 *main.h* 中的宏定义 `#define USE_DHCP` 被注释时，会为 STM32 微控制器分配静态 IP 地址 （默认为 192.168.0.10，可在 “main.h” 文件中修改此值）。

若去掉宏 `#define USE_DHCP` 的注释，则启用 DHCP 协议，STM32 将作为 DHCP 客户端

A.2 当以太网网线断开时，应用如何处理？

当网线断开时，以太网外设停止收发业务，网络接口关闭。如果使用了 LCD 控制器，则会显示一条消息，告知用户网线未连接，否则评估板上的红色 LED 灯会亮起。

当用户重新链接网线时，以太网外设再次工作，网络接口建立。如果使用了 LCD 控制器，则会显示一条消息，告知用户静态或动态分配的新IP地址，否则评估板上的黄色LED灯会亮起。

A.3 怎样将应用移植到不同的硬件上？

当使用另一个硬件平台时，请检查 *HAL_ETH_MspInit()* 函数中以太网外设的 GPIO 配置，若应用需要更多 PPP 外设，则还需检查 *HAL_PPP_MspInit()* 或 *HAL_MspInit()*。

9 修订历史

表 12. 文档修订历史

日期	修订	变更
2014 年 3 月 28 日	1	初始版本。

请仔细阅读：

中文翻译仅为方便阅读之目的。该翻译也许不是对本文档最新版本的翻译，如有任何不同，以最新版本的英文原版文档为准。

本文档中信息的提供仅与 ST 产品有关。意法半导体公司及其子公司 （“ST”）保留随时对本文档及本文所述产品与服务进行变更、更正、修改或改进的权利，恕不另行通知。

所有 ST 产品均根据 ST 的销售条款出售。

买方自行负责对本文所述 ST 产品和服务的选择和使用，ST 概不承担与选择或使用本文所述 ST 产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为 ST 授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在 ST 的销售条款中另有说明，否则，ST 对 ST 产品的使用和 / 或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途 （及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

意法半导体的产品不得应用于武器。此外，意法半导体产品也不是为下列用途而设计并不得应用于下列用途：（A）对安全性有特别要求的应用，例如，生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）汽车应用或汽车环境，且 / 或 （D）航天应用或航天环境。如果意法半导体产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向意法半导体发出了书面通知，采购商仍将独自承担因此而导致的任何风险，意法半导体的产品设计规格明确指定的汽车、汽车安全或医疗工业领域专用产品除外。根据相关政府主管部门的规定，ESCC、QML 或 JAN 正式认证产品适用于航天应用。

经销的 ST 产品如有不同于本文档中提出的声明和 / 或技术特点的规定，将立即导致 ST 针对本文所述 ST 产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大 ST 的任何责任。

ST 和 ST 徽标是 ST 在各个国家或地区的商标或注册商标。

本文档中的信息取代之前提供的所有信息。

ST 徽标是意法半导体公司的注册商标。其他所有名称是其各自所有者的财产。

© 2014 STMicroelectronics 保留所有权利

意法半导体集团公司

澳大利亚 - 比利时 - 巴西 - 加拿大 - 中国 - 捷克共和国 - 芬兰 - 法国 - 德国 - 中国香港 - 印度 - 以色列 - 意大利 - 日本 - 马来西亚 - 马耳他 - 摩洛哥 - 菲律宾 - 新加坡 - 西班牙 - 瑞典 - 瑞士 - 英国 - 美国

www.st.com