

LwIP协议栈移植笔记

LwIP协议栈移植笔记

LwIP简介

移植接口解析

移植LwIP协议栈

low_level_init移植接口实现

low_level_output移植接口实现

low_level_input移植接口实现

ENET_IRQHandler中断服务函数实现

platform参数及函数实现

样例说明

lwip_tcp_client

lwip_tcp_server

lwip_udp_client

lwip_udp_server

结语

注意事项

LwIP简介

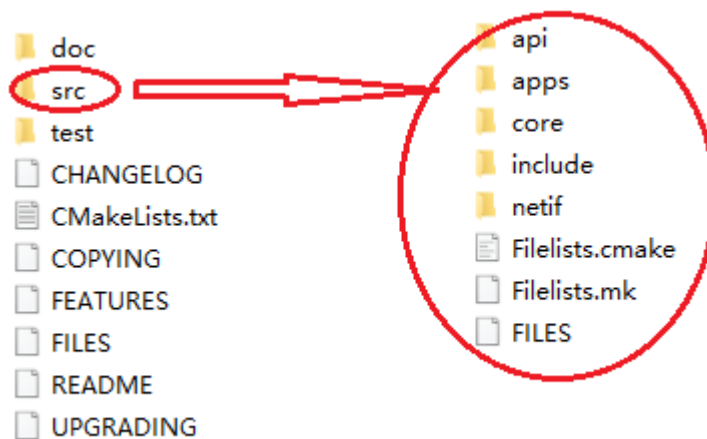
LwIP是轻量化的TCP/IP协议，由瑞典计算机科学院(SICS)的Adam Dunkels 开发的一个小型开源的TCP/IP协议栈。LwIP具有高度可移植性、代码开源，提供了三种编程接口（API）：RAW API、NETCONN API 和 Socket API，用于与TCP/IP代码进行通信。

通过官网（<http://savannah.nongnu.org/projects/lwip/>）可获取LwIP源码包及contrib包。源代码包主要包含LwIP内核的源码文件，contrib包中包含部分移植和应用LwIP的demo。contrib包不属于LwIP内核的一部分，但很有参考价值。

以lwip-2.1.2版本的源码包为例，如图x所示，该源码包分为三部分，`src` 文件为LWIP源代码文件，`doc` 文件包含LwIP相关文档，`test` 为LwIP测试文件，使用时主要关注于 `src` 文件下的内容。

LwIP内核是由一系列模块组合而成，包括 TCP/IP 协议栈的各种协议、内存管理、数据包管理、网卡接口、基础功能类模块、API等，构成这些模块的源文件就分布在api、apps、core、netif中，头文件则汇总在include中。

- api: NETCONN API和Socket API相关的源文件，只有在操作系统的环境中，才能被编译
- apps: 应用程序的源文件，包括常见的应用程序，如httpd、mqtt、tftp、snmp等
- core: LwIP的内核源文件
- include: LwIP所有模块对应的头文件
- netif: 与网卡移植有关的文件



LwIP-2.1.2源码包

移植接口解析

LwIP使用数据结构体netif来描述网卡，并提供统一接口，需要与以太网底层驱动接口函数结合使用，例如底层驱动负责完成网卡的初始化、网卡的数据收发等，当LwIP内核需要发送一个数据包时，就会通过LwIP提供的接口函数去调用底层网卡的发送函数，将数据由硬件接口与软件内核衔接在一起。

contrib文件中包含部分可使用的网卡移植模板文件，其中ethernetif.c文件（contrib-2.1.0\examples\ethernetif目录下的ethernetif.c文件）为底层接口驱动的模板，以LibSamples为例，若要基于LibSample的以太网驱动移植LwIP，则需参考ethernetif.c模板，根据以太网驱动及所需配置进行修改，将底层驱动ethernet相关函数填充到LwIP所需的指定功能函数中。

ethernetif.c文件中的函数通常为与硬件打交道的底层函数，当有数据需要通过网卡接收或者发送数据的时候就会被调用，经过LwIP协议栈内部进行处理后，从应用层就能得到数据或者可以发送数据。该文件中包括函数：low_level_init()、low_level_output()、low_level_input()、ethernetif_input()和ethernetif_init()函数。

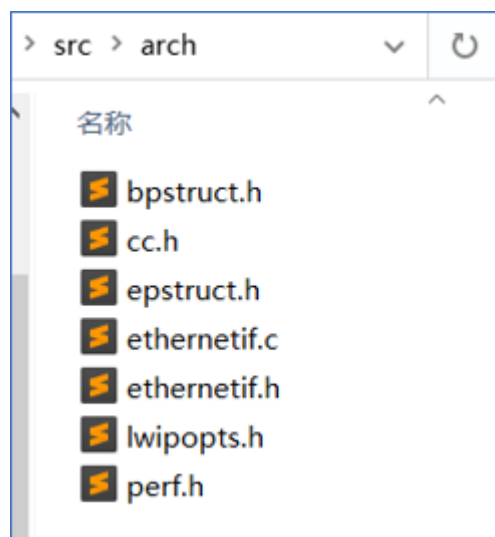
- ethernetif_init()
LwIP中默认的网卡初始化函数，内部封装了low_level_init()函数
- ethernetif_input()
该函数用于接收网卡数据，内部封装了low_level_input()函数，在接收完毕时，将数据通过pbuf递交给上层。
- low_level_init()
low_level_init()函数主要是根据实际情况对网卡进行一系列的初始化工作，例如：初始化MAC地址、长度，设置最大传输包的大小，设置网卡的属性字段等功能。
该函数中需要调用以太网底层驱动中的相关初始化函数，以MindSDK为例，该函数需要调用以太网底层驱动enet_0的PHY、MAC、DMA相关初始化函数并进行配置。
- low_level_output()
该函数用于实现网卡发送数据，是一个底层驱动函数，需根据以太网底层驱动进行相应修改，若想通过一个网卡发送数据，则需要将该数据传入LwIP内核中，经过层层封装最后存储在pbuf数据包中，需注意pbuf以链表的形式存在，数据发送时是以一整个数据包全部发送的。
- low_level_input()
low_level_input()函数用于从网卡中接收一个数据包，并将该数据包封装在pbuf中递交给上层，该函数需要调用以太网底层驱动中的接收函数。

移植LwIP协议栈

基于LibSamples的以太网驱动对LwIP进行移植，需先将LwIP源文件中的部分文件添加到LibSamples中，如：`src`源文件、`include`头文件。

若想令LwIP运行，还需补充contrib文件中部分内容，如图x所示，由于部分源文件中使用头文件写法为“arch/xx”，因此，在src文件下新建arch文件，并将需要修改的模板文件及contrib中的部分接口文件放入arch文件中。

- ethernetif.c网卡移植模板文件
- cc.h文件主要完成协议栈内部使用的数据类型的定义
- lwipopts.h文件包含了用户对协议栈内核参数进行的配置，若未在lwipopts.h文件中进行配置，则LwIP会使用opt.h中的默认参数
- perf.h文件是实现与系通通计和测量相关的功能，若未使用该功能，则无需修改
- bpstruct.h、epstruct.h由contrib文件下的ports文件所提供，属于堆栈的一部分，无需修改



图x LWIP移植所需部分文件

lwipopts.h文件中需要根据是否为操作系统模拟层、堆内存大小、是否使用TCP及TCP相关配置等进行宏定义配置，例如：宏定义 `NO_SYS` 表示无操作系统模拟层，因为当前为无操作系统的移植，所以设置该宏定义为1。

```
1 ...
2 /**
3  * NO_SYS==1: Provides VERY minimal functionality. Otherwise,
4  * use LwIPfacilities.
5  */
6 #define NO_SYS 1
7 ...
```

cc.h文件中包含处理器相关的变量类型、数据结构及字节对齐的相关宏，需根据处理器及编译器进行修改。

```
1 ...
2 #define LWIP_NO_STDINT_H 1
3
4 typedef unsigned char u8_t;
5 typedef signed char s8_t;
6 typedef unsigned short u16_t;
```

```

7  typedef signed    short    s16_t;
8  typedef unsigned  long     u32_t;
9  typedef signed    long     s32_t;
10 typedef u32_t mem_ptr_t;
11 typedef int sys_prot_t;
12
13 #define U16_F "hu"
14 #define S16_F "d"
15 #define X16_F "hx"
16 #define U32_F "u"
17 #define S32_F "d"
18 #define X32_F "x"
19 #define SZT_F "uz"
20 ...
21 #elif defined (__GNUC__)
22
23 #define PACK_STRUCT_BEGIN
24 #define PACK_STRUCT_STRUCT __attribute__((__packed__))
25 #define PACK_STRUCT_END
26 #define PACK_STRUCT_FIELD(x) x
27 ...

```

low_level_init移植接口实现

头文件配置并修改完成后，需要对移植模板文件 ethernetif.c 进行修改。

在以太网底层驱动与LwIP初始化接口的衔接上，对low_level_init()进行修改，在对LwIP的netif结构体进行相关配置之前，需要通过以太网底层驱动使硬件被初始化；初始化后，配置 MAC 硬件地址，链接发送描述符及接收描述符并进行描述符内容配置，配置描述符地址，配置完成后，使能以太网 DMA 启动传输，此时，初始化完成。

```

1  static void
2  low_level_init(struct netif *netif)
3  {
4      struct ethernetif *ethernetif = netif->state;
5
6      /* set MAC hardware address length */
7      netif->hwaddr_len = ETHARP_HWADDR_LEN;
8
9      /* set MAC hardware address */
10     netif->hwaddr[0] = BOARD_MAC_ADDR0;
11     netif->hwaddr[1] = BOARD_MAC_ADDR1;
12     netif->hwaddr[2] = BOARD_MAC_ADDR2;
13     netif->hwaddr[3] = BOARD_MAC_ADDR3;
14     netif->hwaddr[4] = BOARD_MAC_ADDR4;
15     netif->hwaddr[5] = BOARD_MAC_ADDR5;
16
17     /* maximum transfer unit */
18     netif->mtu = 1500;
19
20     /* device capabilities */
21     /* don't set NETIF_FLAG_ETHARP if this device is not an ethernet one */
22     netif->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP |
23     NETIF_FLAG_LINK_UP;

```

```

24 #if LWIP_IPV6 && LWIP_IPV6_MLD
25 /*
26  * For hardware/netifs that implement MAC filtering.
27  * All-nodes link-local is handled by default, so we must let the hardware
    know
28  * to allow multicast packets in.
29  * Should set mld_mac_filter previously. */
30 if (netif->mld_mac_filter != NULL) {
31     ip6_addr_t ip6_allnodes_ll;
32     ip6_addr_set_allnodes_linklocal(&ip6_allnodes_ll);
33     netif->mld_mac_filter(netif, &ip6_allnodes_ll, NETIF_ADD_MAC_FILTER);
34 }
35 #endif /* LWIP_IPV6 && LWIP_IPV6_MLD */
36
37     ETH_GPIOInit();
38     SysTick->CTRL |= ((uint32_t)0x00000004);
39     SysTick_Config(120000000 / 1000);
40
41     ETH_InitTypeDef ptr;
42     ETH_StructInit(&ptr);
43     ptr.ETH_AutoNegotiation = ETH_AutoNegotiation_Disable;
44     ETH_Init(&ptr, ENET_PHY_ADDR);
45     ETH->DMAOMR &= ~ETH_DMAOMR_OSF;
46
47     /* Enable ETH DMA interrupt. */
48     ETH_DMAITConfig(ETH_DMA_IT_NIS|ETH_DMA_IT_R, ENABLE);
49
50     NVIC_InitTypeDef      NVIC_InitStruct;
51     NVIC_InitStruct.NVIC_IRQChannel = ENET_IRQn;
52     NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
53     NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
54     NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
55     NVIC_Init(&NVIC_InitStruct);
56
57     /* Config macd filter address. */
58     ENET_SetupMacAddrFilter(0x1u<<31|0x1u<<5, ENET_ADDR_FILTER_NUM, 0u,
    netif->hwaddr);
59
60     /* Set tx dma desp link. */
61     memset(enet_txdma_desp_tbl, 0, sizeof(enet_txdma_desp_tbl));
62     for (uint32_t i = 0u; i < ENET_TX_NUM - 1; i++) {
63         enet_txdma_desp_tbl[0].CS |= TXDMA_DES0_TCH; /* TCH = 1u. */
64         enet_txdma_desp_tbl[0].BUF1ADDR = (uint32_t)(enet_txbuf[i]);
65         enet_txdma_desp_tbl[0].BUF2NDADDR = (uint32_t>(&enet_txdma_desp_tbl[i
    + 1]));
66     }
67     enet_txdma_desp_tbl[0].CS |= TXDMA_DES0_TCH; /* TCH = 1u. */
68     enet_txdma_desp_tbl[0].BUF1ADDR = (uint32_t)(enet_txbuf[ENET_TX_NUM -
    1]);
69     enet_txdma_desp_tbl[0].BUF2NDADDR = (uint32_t>(&enet_txdma_desp_tbl[0]));
70
71     /* Set enet tx dma descriptor first address. */
72     ETH->DMATXDSAR = (uint32_t>(&enet_txdma_desp_tbl[0]));
73     enet_usable_txdma_desp = &enet_txdma_desp_tbl[0];
74
75     /* Set rx dma desp link. */

```

```

76     memset(enet_rxdma_desp_tbl, 0, sizeof(enet_rxdma_desp_tbl));
77     for (uint32_t i = 0; i < ENET_RX_NUM - 1; i++) {
78         enet_rxdma_desp_tbl[i].CS |= RXDMA_DES0_OWN; /* RDES0[OWN] = 1. */
79         enet_rxdma_desp_tbl[i].BL |= RXDMA_DES1_RCH; /* RDES1[RCH] = 1. */
80         enet_rxdma_desp_tbl[i].BL &= ~ RXDMA_DES1_RBS1;
81         enet_rxdma_desp_tbl[i].BL |= ENET_RX_BUFLen; /* RDES1[RBS1] =
ENET_RX_BUFLen. */
82         enet_rxdma_desp_tbl[i].BUF1ADDR = (uint32_t)enet_rxbuf[i];
83         enet_rxdma_desp_tbl[i].BUF2NDADDR = (uint32_t)
(&enet_rxdma_desp_tbl[i+1]);
84     }
85     enet_rxdma_desp_tbl[ENET_RX_NUM - 1].CS |= RXDMA_DES0_OWN; /* RDES0[OWN]
= 1. */
86     enet_rxdma_desp_tbl[ENET_RX_NUM - 1].BL |= RXDMA_DES1_RCH; /* RDES1[RCH]
= 1. */
87     enet_rxdma_desp_tbl[ENET_RX_NUM - 1].BL &= ~ RXDMA_DES1_RBS1;
88     enet_rxdma_desp_tbl[ENET_RX_NUM - 1].BL |= ENET_RX_BUFLen; /* RDES1[RBS1]
= ENET_RX_BUFLen. */
89     enet_rxdma_desp_tbl[ENET_RX_NUM - 1].BUF1ADDR =
(uint32_t)enet_rxbuf[ENET_RX_NUM - 1];
90     enet_rxdma_desp_tbl[ENET_RX_NUM - 1].BUF2NDADDR = (uint32_t)
(&enet_rxdma_desp_tbl[0]);
91
92     ETH->DMARXDSAR = (uint32_t)enet_rxdma_desp_tbl;
93     enet_first_rxdma_desp = &enet_rxdma_desp_tbl[0];
94
95     ETH_Start();
96 }

```

low_level_output移植接口实现

low_level_output()函数与以太网底层驱动的发送功能函数相结合，将LwIP要发送的数据存储到以太网发送描述符中所指定的存储区域中，再对发送描述符进行配置并进行发送。

```

1  static err_t
2  low_level_output(struct netif *netif, struct pbuf *p)
3  {
4      struct ethernetif *ethernetif = netif->state;
5      struct pbuf *q;
6
7      /* Get current destination address. */
8      ETH_DMADESCTypeDef * txdma_desp = enet_usable_txdma_desp;
9
10     if (0u != (txdma_desp->CS & TXDMA_DES0_OWN) ){
11         return ERR_USE;
12     }
13
14     #if ETH_PAD_SIZE
15         pbuf_remove_header(p, ETH_PAD_SIZE); /* drop the padding word */
16     #endif
17
18     uint32_t e_offset = 0; /* record enet module buf offset. */
19     for (q = p; q != NULL; q = q->next) {
20         /* Send the data from the pbuf to the interface, one pbuf at a
21            time. The size of the data in each pbuf is kept in the ->len

```

```

22     variable. */
23     for (uint32_t i = 0; i < q->len; i++) {
24         ((uint8_t*)(txdma_desp->BUF1ADDR))[e_offset] = ((uint8_t*)(q->payload))
[i];
25         e_offset++;
26         if (e_offset == ENET_TX_BUFLLEN) {
27             txdma_desp = (ETH_DMADESCTypeDef*)(txdma_desp->BUF2NDADDR);
28             if ((txdma_desp->CS & TXDMA_DES0_OWN) != 0u) {
29                 return ERR_USE;
30             }
31             e_offset = 0;
32         }
33     }
34 }
35
36 if (p->tot_len <= ENET_TX_BUFLLEN) {
37     enet_usable_txdma_desp->CS |= TXDMA_DES0_TFS | TXDMA_DES0_TLS |
TXDMA_DES0_OWN;
38     enet_usable_txdma_desp->BL &= ~0x1FFF;
39     enet_usable_txdma_desp->BL |= p->tot_len; /* TBS1!< Transfer buffer size
1. */
40     enet_usable_txdma_desp = (ETH_DMADESCTypeDef*)enet_usable_txdma_desp-
>BUF2NDADDR;
41 } else {
42     enet_usable_txdma_desp->CS |= TXDMA_DES0_TFS; /* TFS = 1u. */
43     enet_usable_txdma_desp->CS &= ~TXDMA_DES0_TLS; /* TLS = 0u. */
44     enet_usable_txdma_desp->BL &= ~0x1FFF;
45     enet_usable_txdma_desp->BL |= ENET_TX_BUFLLEN; /*!< Transfer buffer size
1. */
46     enet_usable_txdma_desp = (ETH_DMADESCTypeDef*)enet_usable_txdma_desp-
>BUF2NDADDR;
47     for (uint32_t i = ENET_TX_BUFLLEN; i < p->tot_len - ENET_TX_BUFLLEN; i+=
ENET_TX_BUFLLEN) {
48         enet_usable_txdma_desp->CS &= ~TXDMA_DES0_TFS; /* TFS = 0u. */
49         enet_usable_txdma_desp->CS &= ~TXDMA_DES0_TLS; /* TLS = 0u. */
50         enet_usable_txdma_desp->BL &= ~0x1FFF;
51         enet_usable_txdma_desp->BL |= ENET_TX_BUFLLEN;
52         enet_usable_txdma_desp = (ETH_DMADESCTypeDef*)enet_usable_txdma_desp-
>BUF2NDADDR;
53     }
54     enet_usable_txdma_desp = (ETH_DMADESCTypeDef*)enet_usable_txdma_desp-
>BUF2NDADDR;
55     enet_usable_txdma_desp->CS &= ~TXDMA_DES0_TFS; /* TFS = 0u. */
56     enet_usable_txdma_desp->CS |= TXDMA_DES0_TLS; /* TLS = 1u. */
57     enet_usable_txdma_desp->BL &= ~0x1FFF;
58     enet_usable_txdma_desp->BL |= (p->tot_len % ENET_TX_BUFLLEN);
59 }
60
61 if (0 != (ETH->DMASR & ETH_DMA_TransmitProcess_Suspended)){
62     ETH_ResumeDMATransmission();
63 }
64
65 MIB2_STATS_NETIF_ADD(netif, ifoutoctets, p->tot_len);
66 if (((u8_t *)p->payload)[0] & 1) {
67     /* broadcast or multicast packet*/
68     MIB2_STATS_NETIF_INC(netif, ifoutnucastpkts);

```

```

69     } else {
70         /* unicast packet */
71         MIB2_STATS_NETIF_INC(netif, ifoutucastpkts);
72     }
73     /* increase ifoutdiscards or ifouterrors on error */
74
75 #if ETH_PAD_SIZE
76     pbuf_add_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
77 #endif
78
79     LINK_STATS_INC(link.xmit);
80
81     return ERR_OK;
82 }

```

low_level_input移植接口实现

low_level_input()函数与以太网底层驱动接收功能函数相结合，将接收到的数据存入LwIP的pbuf链中。ethernetif_input()函数调用low_level_input()函数。

```

1  static struct pbuf *
2  low_level_input(struct netif *netif)
3  {
4      struct ethernetif *ethernetif = netif->state;
5      struct pbuf *p, *q;
6      u16_t len;
7
8      ETH_DMADESCTypeDef * rxdma_desp = enet_first_rxdma_desp;
9      for (uint32_t i = 0; i < ENET_RX_NUM; i++) {
10         if ((rxdma_desp->CS & RXDMA_DES0_RLS) != 0) {
11             len = (uint32_t)(rxdma_desp->CS & RXDMA_DES0_FL)>>16;
12             break;
13         } else if ((rxdma_desp->CS & RXDMA_DES0_OWN) != 0) {
14             return NULL;
15         } else {
16             rxdma_desp = (ETH_DMADESCTypeDef*)(rxdma_desp->BUF2NDADDR);
17         }
18     }
19
20 #if ETH_PAD_SIZE
21     len += ETH_PAD_SIZE; /* allow room for Ethernet padding */
22 #endif
23
24     /* we allocate a pbuf chain of pbufs from the pool. */
25     p = pbuf_alloc(PBUF_RAW, len, PBUF_POOL);
26
27     if (p != NULL) {
28
29 #if ETH_PAD_SIZE
30         pbuf_remove_header(p, ETH_PAD_SIZE); /* drop the padding word */
31 #endif
32
33         /* we iterate over the pbuf chain until we have read the entire
34          * packet into the pbuf. */
35         uint32_t e_offset = 0;

```



```

36     rxdma_desp = enet_first_rxdma_desp;
37     for (q = p; q != NULL; q = q->next) {
38         /* Read enough bytes to fill this pbuf in the chain. The
39          * available data in the pbuf is given by the q->len
40          * variable.
41          * This does not necessarily have to be a memcpy, you can also
preallocate
42          * pbufs for a DMA-enabled MAC and after receiving truncate it to the
43          * actually received size. In this case, ensure the tot_len member of
the
44          * pbuf is the sum of the chained pbuf len members.
45          */
46         for (uint32_t i = 0; i < q->len; i++) {
47             ((uint8_t*)q->payload)[i] = ((uint8_t*)rxdma_desp->BUF1ADDR)
[e_offset];
48             e_offset++;
49             if (e_offset == ENET_RX_BUFLen) {
50                 rxdma_desp = (ETH_DMADESCTypeDef*)(rxdma_desp->BUF2NDADDR);
51                 e_offset = 0;
52             }
53         }
54     }
55
56     MIB2_STATS_NETIF_ADD(netif, ifinoctets, p->tot_len);
57     if (((u8_t *)p->payload)[0] & 1) {
58         /* broadcast or multicast packet*/
59         MIB2_STATS_NETIF_INC(netif, ifinnucastpkts);
60     } else {
61         /* unicast packet*/
62         MIB2_STATS_NETIF_INC(netif, ifinucastpkts);
63     }
64 #if ETH_PAD_SIZE
65     pbuf_add_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
66 #endif
67
68     LINK_STATS_INC(link.recv);
69 } else {
70     LINK_STATS_INC(link.memerr);
71     LINK_STATS_INC(link.drop);
72     MIB2_STATS_NETIF_INC(netif, ifindiscards);
73 }
74
75 do {
76     enet_first_rxdma_desp->CS |= RXDMA_DES0_OWN; /* Set OWN bit. */
77     enet_first_rxdma_desp = (ETH_DMADESCTypeDef*)enet_first_rxdma_desp->
>BUF2NDADDR;
78 } while ((enet_first_rxdma_desp->CS&RXDMA_DES0_OWN) == 0);
79
80 if (RESET != (ETH_GetDMAFlagStatus((0x4 << 17)) ) ){ /*!< ENET dma rx fifo
not active, need to be weak up. */
81     ETH_ResumeDMAReception(); /* wakeup enet dma receive. */
82 }
83
84 return p;
85 }

```

ENET_IRQHandler中断服务函数实现

```
1  /* ENET IRQHandler. */
2  void ENET_IRQHandler()
3  {
4      if (0 != ETH_GetDMAFlagStatus(ETH_DMA_FLAG_R))
5      {
6          ethernetif_input(gnetif);
7          ETH_DMAClearFlag(ETH_DMA_FLAG_R);
8      }
9  }
```

platform参数及函数实现

platform.h 文件中需要根据实际选用的开发板和运行参数等进行宏定义配置，如 IP 地址、端口号、MAC 地址需要根据实际的网络环境进行配置。

```
1  /* initialization enet. */
2  #define ENET_PHY_ADDR          0x00      /* Select PHY address. */
3  #define ENET_PHY_CONTROLREG    0u        /* PHY control register
4  address. */
5  #define ENET_PHY_STATUSREG     1u        /* PHY status register address.
6  */
7  #define ENET_PHY_RESET         0x8000    /* Set PHY reset, use in
8  ENET_PHY_CR registers */
9  #define ENET_PHY_SPEED100M     0x2000    /* Set PHY speed. */
10 #define ENET_PHY_FULLDUPLEX    0x0100    /* Set PHY duplex mode about
11 full duplex. */
12 #define ENET_PHY_LINK          0x0004    /* PHY link-up. */
13 #define ENET_PHY_UNIDIRECTIONAL 0x0080    /* PHY has the ability to
14 encode and transmit data from PHY through MII interface, regardless of
15 whether PHY has determined that an effective link has been connected and
16 established. */
17 #define ENET_PHY_AUTONEGOTIATION 0x1000    /* PHY auto negotiation. */
18 #define ENET_TX_BUFLen        1518u      /* Tx buffer length. */
19 #define ENET_TX_NUM            5u         /* The number of tx. */
20 #define ENET_RX_BUFLen        1518u      /* Configure the frame length
21 of a received frame. */
22 #define ENET_RX_NUM            5u         /* The configured number of
23 received descriptor that can be used for receiving. */
24 #define ENET_ADDR_FILTER_NUM    1u        /* Select MAC address filter
25 number from 0~5. */
26
27 #define BOARD_MAC_ADDR0        2u
28 #define BOARD_MAC_ADDR1        0u
29 #define BOARD_MAC_ADDR2        0u
30 #define BOARD_MAC_ADDR3        0u
31 #define BOARD_MAC_ADDR4        0u
32 #define BOARD_MAC_ADDR5        0u
33
34 #define BOARD_IP_ADDR0         169u
35 #define BOARD_IP_ADDR1         254u
36 #define BOARD_IP_ADDR2         102u
37 #define BOARD_IP_ADDR3         101u
```

```

28
29 #define BOARD_NETMASK_ADDR0      255u
30 #define BOARD_NETMASK_ADDR1      255u
31 #define BOARD_NETMASK_ADDR2      255u
32 #define BOARD_NETMASK_ADDR3      0u
33
34 #define BOARD_GW_ADDR0            192u
35 #define BOARD_GW_ADDR1            168u
36 #define BOARD_GW_ADDR2            1u
37 #define BOARD_GW_ADDR3            1u
38
39 #define BOARD_UDP_OWN_PORT        6800u
40
41 #define TXDMA_DES0_TCH             0x01u<<20
42 #define TXDMA_DES0_TFS             0x01u<<28
43 #define TXDMA_DES0_TLS             0x01u<<29
44 #define TXDMA_DES0_OWN             0x01u<<31
45
46 #define RXDMA_DES0_RLS             0x01u<<8
47 #define RXDMA_DES0_FL              0x3FFFu<<16
48 #define RXDMA_DES0_OWN             0x01u<<31
49 #define RXDMA_DES1_RCH             0x01u<<14
50 #define RXDMA_DES1_RBS1            0x1FFFu
51
52 EXTERN volatile uint32_t systime_ms;

```

platform.c文件中除了对Ethernet相关的时钟引脚进行配置及使用到的系统时钟对应参数申明外，也根据LwIP协议栈实际的应用需求，实现了关于MAC地址过滤器的函数。

```

1 void ENET_SetupMacAddrFilter(uint32_t filter, uint32_t addr_id, uint32_t
  addr_mask, uint8_t * addr)
2 {
3     ETH->MACAFR |= filter;
4
5     if ( (0u != (filter & ETH_SourceAddrFilter_Normal_Enable)) || (0u !=
  (filter & 0x100)) ) /* Set source address filter. */
6     {
7         ETH->MACA0HR = ( 0x1u<<31 | 0x1u<<30 | (uint32_t)addr[4u] |
  ((uint32_t)addr[5u]<<8u) );
8         ETH->MACA0LR = ( (uint32_t)addr[0u] | ((uint32_t)addr[1u] << 8u) |
  ((uint32_t)addr[2u] << 16u) | ((uint32_t)addr[3u] << 24u) );
9     }
10    else if ( (0u != (filter & 0x10)) || (0u != (filter & 0x100)) ) /* Set
  destination address filter. */
11    {
12        ETH->MACAFR &= ~(0x1u<<4 | 0x1u<<1);
13    }
14
15    if (0u != addr_mask)
16    {
17        ETH->MACA0HR |= addr_mask;
18    }
19 }

```

```
1  /* Returns the current time in milliseconds, this API from lwip/sys.h */
2  uint32_t sys_now(void)
3  {
4      return systime_ms;
5  }
6
7  uint32_t sys_jiffies(void)
8  {
9      return systime_ms * 1000000;
10 }
```

样例说明

基于移植的 LwIP 协议，LibSamples 还提供了展示 LwIP 中部分协议功能的样例工程，以 PLUS-F5270 V2.0 开发板为例，展示 TCP 协议客户端与服务器通信的 lwip_tcp_client、lwip_tcp_server 样例，展示 UDP 协议客户端与服务器通信的 lwip_udp_client、lwip_udp_server。

样例实现环境搭建

- 本文基于搭载了 MM32F5277E9P MCU 的开发板 PLUS-F5270 V2.0 进行实现，使用 2 根网线，分别连接电脑与路由器、开发板与路由器。
- 在官网(<http://free.cmsoft.cn/reslink.php?id=205>)下载网络调试助手 NetAssist 并安装，用于后续的样例功能验证。
- 打开电脑终端(WIN+R 键，输入 CMD)，然后输入指令 `ipconfig/all`，查看本机的以太网 IP 地址为 `192.168.108.85`，
- 在终端中输入命令 `netstat -na` 获取本地开放端口，这里我们获取到可用端口号为 `49153`。

lwip_tcp_client


lwip_tcp_client 样例用于展示基于以太网及 LwIP 使用 TCP 协议作为客户端，进行客户端与服务器之间的通信。

若想使用 LwIP，则需要先将协议栈初始化，并设置主机的 IP 地址、子网掩码、网关地址等。需注意，样例工程中所设置的 IP 地址需要与路由器处于同一子网，如图 x 所示，在命令提示符 (CMD) 中使用命令 `ipconfig/all` 可查看各 IP 的详细信息，例如所查出的以太网 IPx4 地址为 192.168.108.85，则在样例工程中可设置 IP 地址为 192.168.108.98，网关地址与子网掩码的配置需与所查出的以太网默认网关及子网掩码相同。

```

1 void app_lwip_init(void)
2 {
3     ip4_addr_t ipaddr;
4     ip4_addr_t netmask;
5     ip4_addr_t gw;
6     IP4_ADDR(&ipaddr, BOARD_IP_ADDR0, BOARD_IP_ADDR1, BOARD_IP_ADDR2,
BOARD_IP_ADDR3);
7     IP4_ADDR(&netmask, BOARD_NETMASK_ADDR0, BOARD_NETMASK_ADDR1,
BOARD_NETMASK_ADDR2, BOARD_NETMASK_ADDR3);
8     IP4_ADDR(&gw, BOARD_GW_ADDR0, BOARD_GW_ADDR1, BOARD_GW_ADDR2,
BOARD_GW_ADDR3);
9     lwip_init();
10    ...
11 }

```

 命令提示符

```

(c) Microsoft Corporation。保留所有权利。
C:\Users\MindMotion>ipconfig/all

Windows IP 配置

主机名 . . . . . : DESKTOP-MCR7G5A
主 DNS 后缀 . . . . . :
节点类型 . . . . . : 混合
IP 路由已启用 . . . . . : 否
WINS 代理已启用 . . . . . : 否

以太网适配器 以太网 3:

媒体状态 . . . . . : 媒体已断开连接
连接特定的 DNS 后缀 . . . . . :
描述. . . . . : ASIX AX88179A USB 3.2 Gen1 to Gigabit Ethernet Adapter
物理地址. . . . . : F8-E4-3B-74-31-7B
DHCP 已启用 . . . . . : 是
自动配置已启用. . . . . : 是

以太网适配器 以太网:

连接特定的 DNS 后缀 . . . . . :
描述. . . . . : Realtek PCIe GbE Family Controller
物理地址. . . . . : 38-F3-AB-4E-3E-A9
DHCP 已启用 . . . . . : 是
自动配置已启用. . . . . : 是
本地链接 IPv6 地址. . . . . : fe80::6f59:3e34:49c8:e0e%7 (首选)
IPv4 地址 . . . . . : 192.168.105.85 (首选)
子网掩码 . . . . . : 255.255.255.0
获得租约的时间 . . . . . : 2023年7月10日 9:00:02
租约过期的时间 . . . . . : 2023年7月12日 9:14:59
默认网关. . . . . : 192.168.105.1
DHCP 服务器 . . . . . : 192.168.105.1
DHCPv6 IAID . . . . . : 104395691
DHCPv6 客户端 DUID . . . . . : 00-01-00-01-28-10-E6-24-38-F3-AB-4E-3E-A9
DNS 服务器 . . . . . : 202.96.209.133
TCP/IP 上的 NetBIOS . . . . . : 已启用

```

图x 在CMD界面通过命令查询以太网IP信息

在配置完IP地址等必要信息后，需挂载网卡，在LWIP中网卡挂载函数为 `netif_add()` 函数，将所配置的数据传入该函数中。

```

1 void app_lwip_init(void)
2 {
3     ...
4     netif_add(&gnetif, &ipaddr, &netmask, &gw, NULL, &ethernetif_init,
&ethernet_input);
5
6     netif_set_default(&gnetif);

```

```

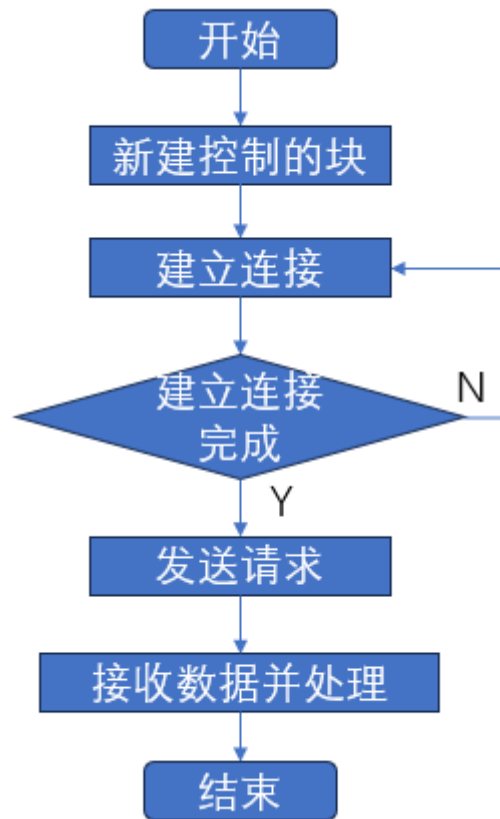
7
8     if (netif_is_link_up(&gnetif))
9     {
10         netif_set_up(&gnetif);
11     }
12     else
13     {
14         netif_set_down(&gnetif);
15     }
16 }

```

Lwip协议栈初始化后，需要对所使用的 TCP Client（TCP客户端）进行初始化配置。在 LwIP中存在多个与 TCP 相关的函数，lwip_tcp_client 样例所使用到的函数包括：

- tcp_new()
创建一个TCP的PCB控制块
- tcp_connect()
连接远端主机
- tcp_err()
控制块err字段注册的回调函数，遇到错误时被调用
- tcp_write()
构造一个报文并放入控制块的发送缓冲队列中
- tcp_recv()
控制块rev字段注册的回调函数，当接收到新数据是被调用
- tcp_recved()
当程序处理完数据后调用该函数，通知内核更新接收窗口
- tcp_close()
关闭一个TCP连接

TCP 客户端的工作流程包括：新建控制块、建立连接、发送请求与接收数据并处理。TCP客户端工作流程如图x所示。



图x TCP客户端流程图

TCP Client (TCP客户端) 进行初始化配置时, 通过 IP4_ADDR() 函数将目标服务器的IP写入结构体; 再通过 tcp_new() 函数为TCP客户端分配一个结构, 当该结构不为空时, 使用 tcp_connect() 函数与目标服务器进行连接, 该函数中配置目标端口和目标IP参数并调用连接完成回调函数。

```

1 void app_tcp_client_init(void)
2 {
3     struct tcp_pcb *tcp_client_pcb;
4     ip_addr_t app_server_ip;
5     /* Write the IP of the target server into a structure, which is the local
6      connection IP address of the pc. */
7     IP4_ADDR(&app_server_ip, BOARD_TCP_SERVER_IPADDR0,
8 BOARD_TCP_SERVER_IPADDR1, BOARD_TCP_SERVER_IPADDR2,
9 BOARD_TCP_SERVER_IPADDR3);
10    /* Assign a structure to the TCP client */
11    tcp_client_pcb = tcp_new();
12
13    if (tcp_client_pcb != NULL)
14    {
15        /* Connect with the target server, and the parameters include the
16        target port and the target IP. */
17        tcp_connect(tcp_client_pcb, &app_server_ip, BOARD_TCP_SERVER_PORT,
18 app_tcp_client_connected);
19        /* Registered connection error handling callback function. */
20        tcp_err(tcp_client_pcb, app_tcp_client_connecterror);
21    }
22 }

```

在连接完成回调函数中, 使用 tcp_write() 函数发送问候字符串以建立连接, 并使用 tcp_recv() 函数配置接收回调函数。

```

1 static err_t app_tcp_client_connected(void *arg, struct tcp_pcb *pcb, err_t
  err)
2 {
3     /* Send a greeting string to establish a connection */
4     tcp_write(pcb, clientstring, strlen(clientstring), 1u);
5     /* Configure the receive callback function */
6     tcp_recv(pcb, app_tcp_client_xfer);
7
8     return ERR_OK;
9 }

```

在TCP客户端接收数据后的数据处理回调函数中，接收到有效数据后，通过tcp_recved()更新接收窗口，使用 tcp_write() 函数将接收到的服务器内容回显。

```

1 static err_t app_tcp_client_xfer(void *arg, struct tcp_pcb *pcb, struct pbuf
  *tcp_recv_pbuf, err_t err)
2 {
3     if (tcp_recv_pbuf != NULL)
4     {
5         /* Update the receiving window */
6         tcp_recved(pcb, tcp_recv_pbuf->tot_len);
7         tcp_write(pcb, tcp_recv_pbuf->payload, tcp_recv_pbuf->len, 1u);
8
9         pbuf_free(tcp_recv_pbuf);
10    }
11    else if (err == ERR_OK)
12    {
13        tcp_close(pcb);
14        app_tcp_client_init();
15        return ERR_OK;
16    }
17    return ERR_OK;
18 }

```

lwip_tcp_client 样例的实验现象如图x所示，通过网络调试助手可查看到连接成功后，远端服务器收到客户端发送的数据，服务器向客户端发送任意数据包后，客户端回显相同数据。

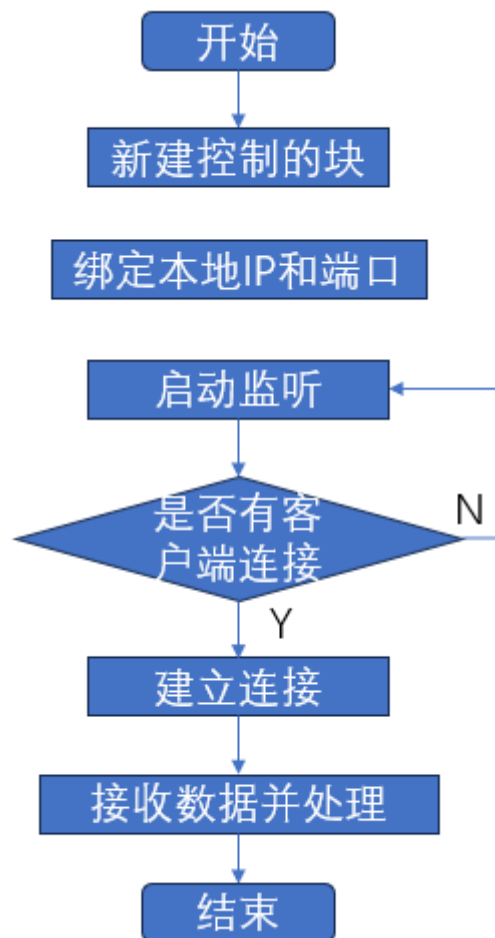


图x lwip_tcp_client样例实验现象

lwip_tcp_server

lwip_tcp_server 样例用于展示基于以太网及 LwIP使用 TCP 协议作为服务器，进行客户端与服务端之间的通信。lwip_tcp_server 样例在 LwIP协议栈初始化部分及设置IP地址、子网掩码、网关地址等参数的配置，与lwip_tcp_client 样例相同。

TCP服务器的工作流程包括：新建控制块、绑定控制块、控制块监听、建立连接以及接收并处理数据，如图x所示。



图x TCP 服务器工作流程

初始化 TCP Server (TCP 服务器)，通过 `tcp_new()` 函数为TCP服务器分配一个结构体，使用 `tcp_bind()` 函数绑定本地端口号及IP地址，设定端口号为 `BOARD_TCP_SERVER_PORT`，IP地址为 `IP_ADDR_ANY`，使用 `tcp_listen()` 函数监听之前创建的结构体，最后通过 `tcp_accept()` 函数初始化结构体接收回调函数。

```
1 void app_tcp_server_init(void)
2 {
3     struct tcp_pcb *pcb;
4     /* Create new control block. */
5     pcb = tcp_new();
6     /* Binding local port and IP address. */
7     tcp_bind(pcb, IP_ADDR_ANY, BOARD_TCP_SERVER_PORT);
8     /* Start monitoring. */
9     pcb = tcp_listen(pcb);
10    /* Initialize the structure to receive the callback function. */
11    tcp_accept(pcb, app_tcp_server_accept);
12 }
```

TCP服务器接收回调函数中，通过 `tcp_recv()` 函数注册接收回调函数，该回调函数中包含TCP服务器接收数据处理，在接收到客户端的数据后，服务器向客户端发送一段字符数据表示已接收到并将接受的数据发送回去，释放pbuf并结束此次TCP连接。

```
1 static err_t app_tcp_server_accept(void *arg, struct tcp_pcb *pcb, err_t
2 err)
3 {
```

```

3     tcp_recv(pcb, app_tcp_server_receive);
4     return ERR_OK;
5 }
6 /* TCP server receive processing. */
7 static err_t app_tcp_server_receive(void *arg, struct tcp_pcb *pcb, struct
pbuf *tcp_recv_pbuf, err_t err)
8 {
9     if (tcp_recv_pbuf != NULL)
10    {
11        /* Update the receiving window */
12        tcp_recved(pcb, tcp_recv_pbuf->tot_len);
13        tcp_write(pcb, echostring, strlen(echostring), 1u);
14        /* send the receive data. */
15        tcp_write(pcb, tcp_recv_pbuf->payload, tcp_recv_pbuf->len, 1u);
16        pbuf_free(tcp_recv_pbuf);
17        tcp_close(pcb); /* close connect. */
18    }
19    else if(err == ERR_OK)
20    {
21        return tcp_close(pcb);
22    }
23    return ERR_OK;
24 }

```

lwip_tcp_server 样例的实验现象如图x所示。网络调试助手做TCP客户端，向开发板所做的服务器发送数据，服务器接收数据后发送一段字符并将该数据发回客户端。



图x lwip_tcp_server实验现象

lwip_udp_client

lwip_udp_client 样例用于展示基于以太网及 LwIP使用 UDP 协议作为客户端，进行客户端与服务端之间的通信。在 LwIP中存在多个与 UDP 相关的函数，例如：

- udp_new
新建一个UDP的PCB块
- udp_remove
将一个PCB控制块从链表中删除，并释放相应的内存
- udp_bind
为UDP的PCB控制块绑定一个本地IP地址和端口号
- udp_connect
连接到指定IP地址主机的指定端口上
- udp_disconnect
断开连接，将控制块设置为非连续状态
- udp_send
通过一个PCB控制块发送数据
- udp_recv
需要创建的一个回调函数，当接收到数据的时候被调用

lwip_udp_client 样例在 LwIP 协议栈初始化部分及设置 IP 地址、子网掩码、网关地址等参数的配置，与 lwip_tcp_client 样例相同。

LWIP 协议栈初始化完成后，进行 UDP 做客户端初始化，通过 IP4_ADDR() 函数设置服务端的 IP 地址，再由 udp_new() 函数创建一个新的 UDP 控制块，通过 udp_bind() 的函数为 UDP 的 PCB 控制块绑定一个本地 IP 地址和端口号，设置端口号为 BOARD_UDP_OWN_PORT。由 udp_connect() 函数连接到指定 IP 地址主机的指定端口，端口为 BOARD_UDP_ECHO_PORT，通过 udp_recv() 函数注册回调函数。

```
1 void app_udp_client_init(void)
2 {
3     ip_addr_t udp_server_addr;
4     struct udp_pcb * pcb;
5     IP4_ADDR(&udp_server_addr, BOARD_UDP_SERVER_IPADDR0,
BOARD_UDP_SERVER_IPADDR1, BOARD_UDP_SERVER_IPADDR2,
BOARD_UDP_SERVER_IPADDR3);
6     pcb = udp_new(); /* Generate a new UDP control block */
7     if (pcb != NULL)
8     {
9         udp_bind(pcb, IP_ADDR_ANY, BOARD_UDP_OWN_PORT);
10        udp_connect(pcb, &udp_server_addr, BOARD_UDP_ECHO_PORT);
11        udp_recv(pcb, app_udp_server_callback, NULL); /* Receive callback.
*/
12        app_udp_client_send(pcb, app_udp_client_data);
13    }
14 }
```

UDP 做客户端先发送自定义字符数据，等待收到来自服务器的数据后，将数据发回服务器。

lwip_udp_client 样例实验现象如图x所示。



图x lwip_udp_client实验现象

lwip_udp_server

lwip_udp_server 样例用于展示基于以太网及 LwIP使用 UDP 协议作为服务器，进行客户端与服务端之间的通信。

lwip_udp_server 样例在 LwIP协议栈初始化部分及设置IP地址、子网掩码、网关地址等参数的配置，与 lwip_udp_client 样例相同。

LWIP协议栈初始化完成后，进行 UDP 做服务器初始化，通过 `udp_new()` 函数生成一个新的UDP控制块，`udp_bind()` 函数绑定任意IP地址 `IP_ADDR_ANY` 及指定端口 `BOARD_UDP_OWN_PORT`，通过 `udp_recv()` 函数注册回调函数。

```

1 void app_udp_server_init(void)
2 {
3     struct udp_pcb * pcb;
4     pcb = udp_new(); /* Generate a new UDP control block */
5     udp_bind(pcb, IP_ADDR_ANY, BOARD_UDP_OWN_PORT); /* Bind upcb block to
6     any IP address and specified port */
7     udp_recv(pcb, app_udp_server_callback, NULL); /* Receive callback. */
8 }

```

UDP服务器回调函数用于发送函数到客户端，通过 `udp_sendto()` 函数发送数据包，`udp_disconnect()` 函数断开链接。

```

1 static void app_udp_server_callback(void *arg, struct udp_pcb *upcb, struct pbuf
  *p, const ip_addr_t *addr, u16_t port)
2 {
3     udp_sendto(upcb, p, addr, port); /* Send data to destination address. */
4     udp_disconnect(upcb);           /* Remove the remote end of the pcb. */
5 }

```

lwip_udp_server 样例的实验现象如图x所示。



图x lwip_udp_server 实验现象

结语

本文对LwIP协议栈进行了简单介绍，并对 TCP 协议与 UDP 协议的客户端与服务器之间的通信进行分析，最后对 LwIP的移植和样例演示操作进行了讲解。

注意事项

- 在配置 IP 地址和端口号时，当连接了WIFI后需要注意我们选用的是以太网的IP地址，而非WLAN的IP地址。
- lwip_tcp_server 样例演示中，在 网络调试助手 工具中的左侧 远程主机地址 参数填写要和修改样例中的 BOARD_IP_ADDR0~3 的参数一致，且要在主机以太网所分配的局域网IP地址范围内。（BOARD_GW_ADDR0~3参数全程可不修改）

- 在裸机运行环境中，使用到了 SysTick 作为时间基准，需要在 platform.c/.h 文件中进行声明和定义，以及在 mm32f5270_it.c 文件中进行中断服务函数的实现。