lwip_tcp_client

简介

LWIP是轻量化的TCP/IP协议，意在用少量的资源消耗实现一个较为完整的TCP/IP协议栈，在保持TCP协议主要功能的基础上减少对RAM的占用。
在向以太网驱动移植LWIP时，会对部分头文件进行修改，并根据LWIP所提供的网卡接口模板文件ethernetif.c将以太网驱动与LWIP相结合。通常，在使用LWIP时，会根据自身需求对部分头文件中的宏进行修改。

LWIP移植

本样例使用无操作系统移植LWIP，使用LWIP文件为lwip-2.1.2版本，可从官网(http://savannah.nongnu.org/projects/lwip/)的Download Area选项获取所需的lwip-2.1.2源码文件及contrib-2.1.0样例包，contrib-2.1.0文件包不属于LWIP内核，而是一个包含移植和应用LWIP的一些应用示例的文件包，官网如图1所示。

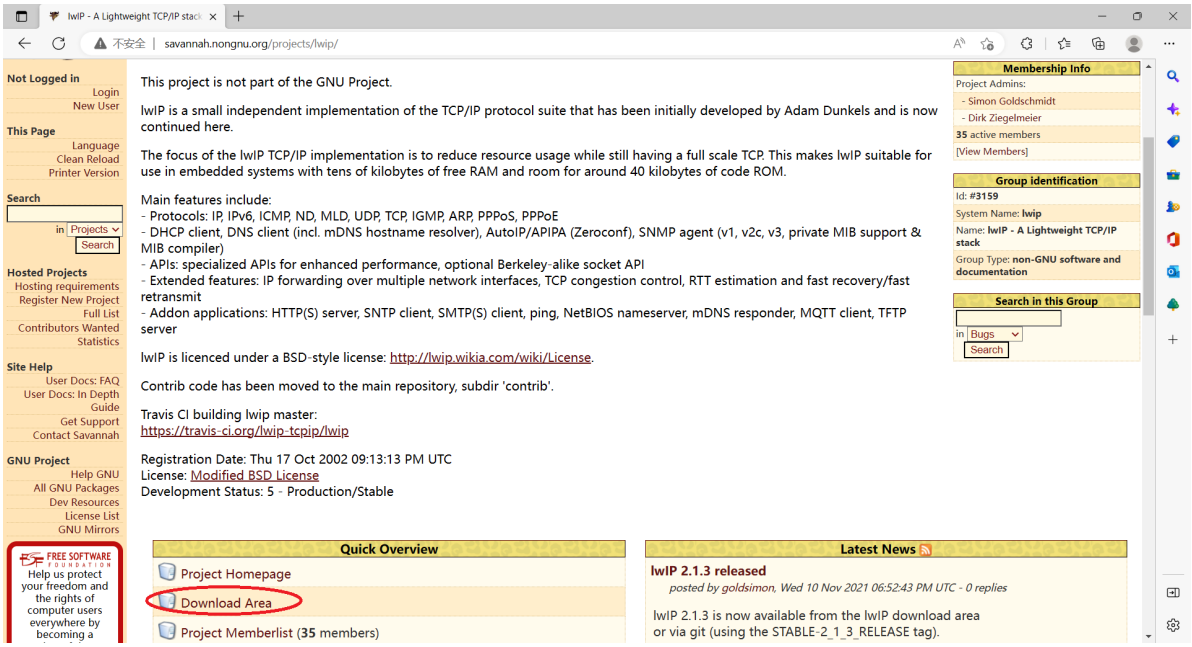

图1 在LWIP官网上获取lwip-2.1.2源码

在lwip-2.1.2中，所需源码存放在`lwip-2.1.2/src`文件夹下，在MindSDK的components文件下新建lwip文件，并将`lwip-2.1.2/src`中的内容存放在此，如图2所示。

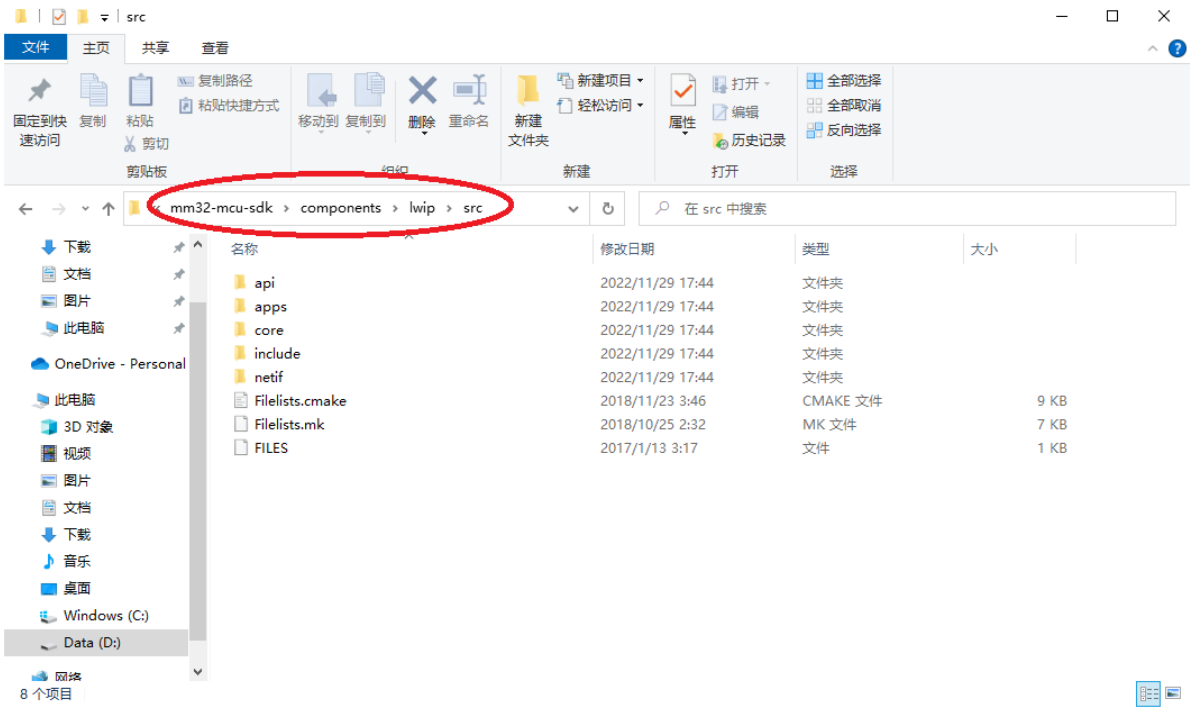

图2 在MindSDK上添加LWIP源码

在添加完源代码后，仍需添加部分与底层接口相关的文件，从contrib-2.1.0文件中获取，在样例中新建`arch`文件用于存放相关文件，其中lwipopts.h、perf.h与cc.h头文件及ethernetif.c文件在移植时需要进行配置修改，arch文件包含内容如图3所示。

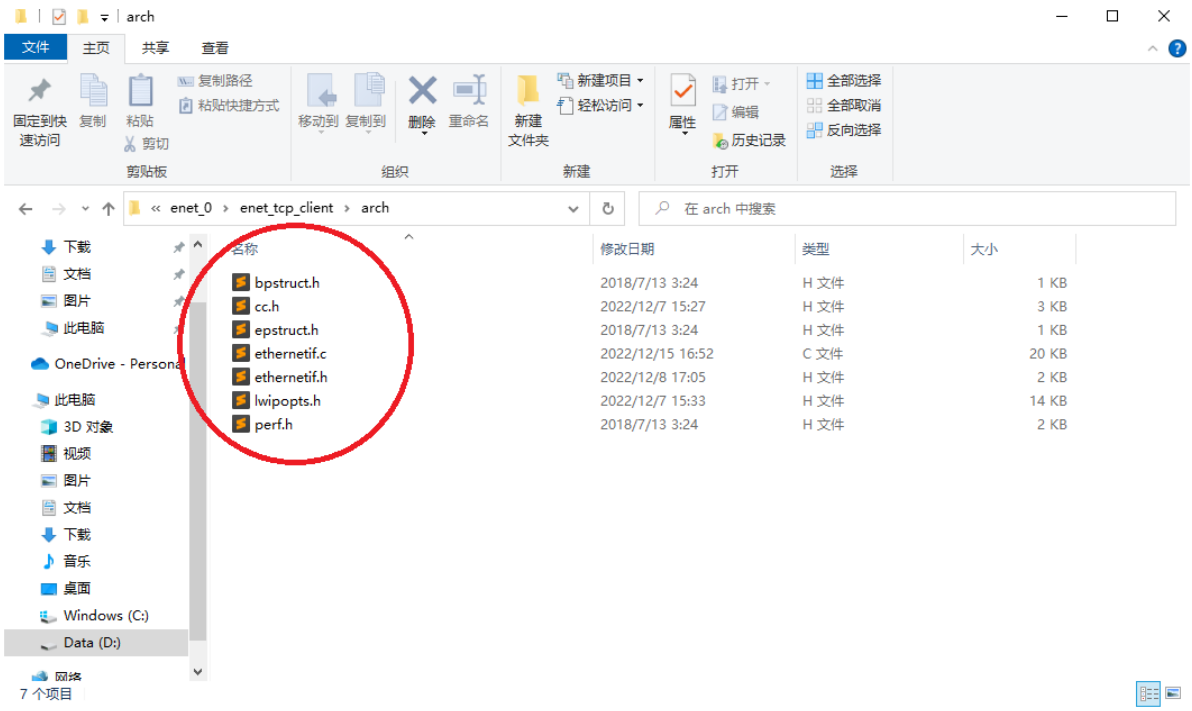

图3 在MindSDK上添加底层接口相关文件

在进行LWIP移植时，需在头文件cc.h、perf.h、lwipopts.h中根据自身配置修改对应参数。

- cc.h文件主要完成协议栈内部使用的数据类型的定义
  该文件需要根据所用处理器及编译器特性进行定义，当前将变量定义为C语言的基本类型。

```
...
typedef unsigned   char    u8_t;
typedef signed     char    s8_t;
typedef unsigned   short   u16_t;
```

```
typedef signed     short   s16_t;
typedef unsigned   long    u32_t;
typedef signed     long    s32_t;
typedef u32_t mem_ptr_t;
typedef int sys_prot_t;
...
#define PACK_STRUCT_BEGIN
#define PACK_STRUCT_STRUCT __attribute__ ((__packed__))
#define PACK_STRUCT_END
#define PACK_STRUCT_FIELD(x) x
...
```

- perf.h是和系统统计与测量相关的头文件
  本样例中未使用任何统计与测量功能，因此，当前perf.h文件未修改。

```
#ifndef LWIP_PERF_H
#define LWIP_PERF_H

#define PERF_START    /* null definition */
#define PERF_STOP(x)  /* null definition */

#endif /* LWIP_PERF_H */
```

- lwipopts.h包含用户对协议栈内核的参数配置
  该文件是将LWIP的opt.h中的部分需要修改的参数配置提取出来，在运行时，若在lwipopts.h与opt.h中都配置了对应的宏定义参数，则参数以lwipopts.h的为准，非共有的参数以各自的配置为准。可对是否使用操作系统、字节对齐、堆内存大小、是否使用TCP等参数进行配置。

```
#define SYS_LIGHTWEIGHT_PROT    0  /* Unprotected critical area. */
#define NO_SYS                  1  /* No operating system. */
#define NO_SYS_NO_TIMERS        0  /* Drop support for sys_timeout when
NO_SYS==1. */
#define MEM_ALIGNMENT           4U /* 4 byte alignment. */
#define MEM_SIZE               (15*1024)  /* The size of the heap memory. */
#define MEMP_NUM_PBUF           25  /* The number of memp struct pbufs */
#define MEMP_NUM_UDP_PCB        4   /* The number of UDP protocol control blocks.
*/
#define MEMP_NUM_TCP_PCB        6   /* The number of simulatenously active TCP
connections. */
#define MEMP_NUM_TCP_PCB_LISTEN 6   /* The number of listening TCP connections.
*/
#define MEMP_NUM_TCP_SEG        150 /* The number of simultaneously queued TCP
segments. */
#define MEMP_NUM_SYS_TIMEOUT    6   /* The number of simulateously active
timeouts. */
#define PBUF_POOL_SIZE          45  /* The number of buffers in the pbuf pool.
*/
#define PBUF_POOL_BUFSIZE
LWIP_MEM_ALIGN_SIZE(TCP_MSS+40+PBUF_LINK_ENCAPSULATION_HLEN+PBUF_LINK_HLEN)  /*
The size of each pbuf in the pbuf pool. */
#define LWIP_TCP                1  /* Enable TCP. */
#define TCP_TTL                 255 /* The Time-To-Live value. of TCP. */
#define TCP_QUEUE_OOSEQ         0  /* Controls if TCP should queue segments that
arrive out of order. */
```

```
#define TCP_MSS                 (1500 - 40)  /* Maximum TCP segment size. */
#define TCP_SND_BUF             (10*TCP_MSS) /* TCP sender buffer space (bytes).
*/
#define TCP_SND_QUEUELEN       (8* TCP_SND_BUF/TCP_MSS)  /* TCP sender buffer
space (pbufs). */
#define TCP_WND                 (11*TCP_MSS) /* The size of a TCP window.  This
must be at least (2 * TCP_MSS) for things to work well. */
#define LWIP_ICMP               1  /* Enable ICMP. */
#define LWIP_DHCP               1  /* Enable DHCP. */
#define LWIP_UDP                1  /* Enable UDP. */
#define UDP_TTL                 255  /* The Time-To-Live value. of UDP. */
#define LWIP_STATS              0  /* Disable statistics collection in
lwip_stats. */
#define LWIP_PROVIDE_ERRNO      1  /* Let lwIP provide ERRNO values and the
'errno' variable. */
#define LWIP_NETIF_LINK_CALLBACK    0 /* Callback function not support interface.
*/

#ifdef CHECKSUM_BY_HARDWARE
  /* CHECKSUM_GEN_IP==0: Generate checksums by hardware for outgoing IP
packets.*/
  #define CHECKSUM_GEN_IP                 0
  /* CHECKSUM_GEN_UDP==0: Generate checksums by hardware for outgoing UDP
packets.*/
  #define CHECKSUM_GEN_UDP                0
  /* CHECKSUM_GEN_TCP==0: Generate checksums by hardware for outgoing TCP
packets.*/
  #define CHECKSUM_GEN_TCP                0
  /* CHECKSUM_CHECK_IP==0: Check checksums by hardware for incoming IP
packets.*/
  #define CHECKSUM_CHECK_IP               0
  /* CHECKSUM_CHECK_UDP==0: Check checksums by hardware for incoming UDP
packets.*/
  #define CHECKSUM_CHECK_UDP              0
  /* CHECKSUM_CHECK_TCP==0: Check checksums by hardware for incoming TCP
packets.*/
  #define CHECKSUM_CHECK_TCP              0
  /* CHECKSUM_CHECK_ICMP==0: Check checksums by hardware for incoming ICMP
packets.*/
  #define CHECKSUM_GEN_ICMP               0
#else
  /* CHECKSUM_GEN_IP==1: Generate checksums in software for outgoing IP
packets.*/
  #define CHECKSUM_GEN_IP                 1
  /* CHECKSUM_GEN_UDP==1: Generate checksums in software for outgoing UDP
packets.*/
  #define CHECKSUM_GEN_UDP                1
  /* CHECKSUM_GEN_TCP==1: Generate checksums in software for outgoing TCP
packets.*/
  #define CHECKSUM_GEN_TCP                1
  /* CHECKSUM_CHECK_IP==1: Check checksums in software for incoming IP
packets.*/
  #define CHECKSUM_CHECK_IP               1
  /* CHECKSUM_CHECK_UDP==1: Check checksums in software for incoming UDP
packets.*/
```

```
  #define CHECKSUM_CHECK_UDP              1
  /* CHECKSUM_CHECK_TCP==1: Check checksums in software for incoming TCP
packets.*/
  #define CHECKSUM_CHECK_TCP              1
  /* CHECKSUM_CHECK_ICMP==1: Check checksums by hardware for incoming ICMP
packets.*/
  #define CHECKSUM_GEN_ICMP               1
#endif
#define LWIP_NETCONN                    0  /* Disable Netconn API. */
#define LWIP_SOCKET                     0  /* Disable Socket API. */
```

- ethernetif.c文件是网卡驱动的框架，使用以太网底层驱动填充该框架即可。
  将MindSDK的以太网驱动与下列函数相结合使用：
  - low_level_init()
    网卡初始化函数，用于网卡的复位及参数初始化，根据实际的网卡属性进行配置netif中与网卡相关的字段，如MAC地址等。
  - low_level_output()
    网卡发送函数，将内核的数据包发送出去，数据包采用pbuf数据结构进行描述。
  - low_level_input()
    网卡接收函数，会接收一个数据包，需将接收的数据封装为pbuf的形式。
  - ethernetif_input()
    该函数调用low_level_input()函数从网卡中读取一个数据包，再解析该数据包类型并将包递交给上层，当前无需修改该函数。
  - ethernetif_init()
    在上层管理网卡netif时会调用该函数，该函数调用low_level_init()函数，由于当前只有一个网卡，故无需对该函数进行改写。

使用指南

1. **使用网线通过路由器连接开发板上的以太网接口与电脑以太网接口**

使用3根网线，分别连接电脑与路由器、路由器与开发板、路由器与网线，如图4所示。

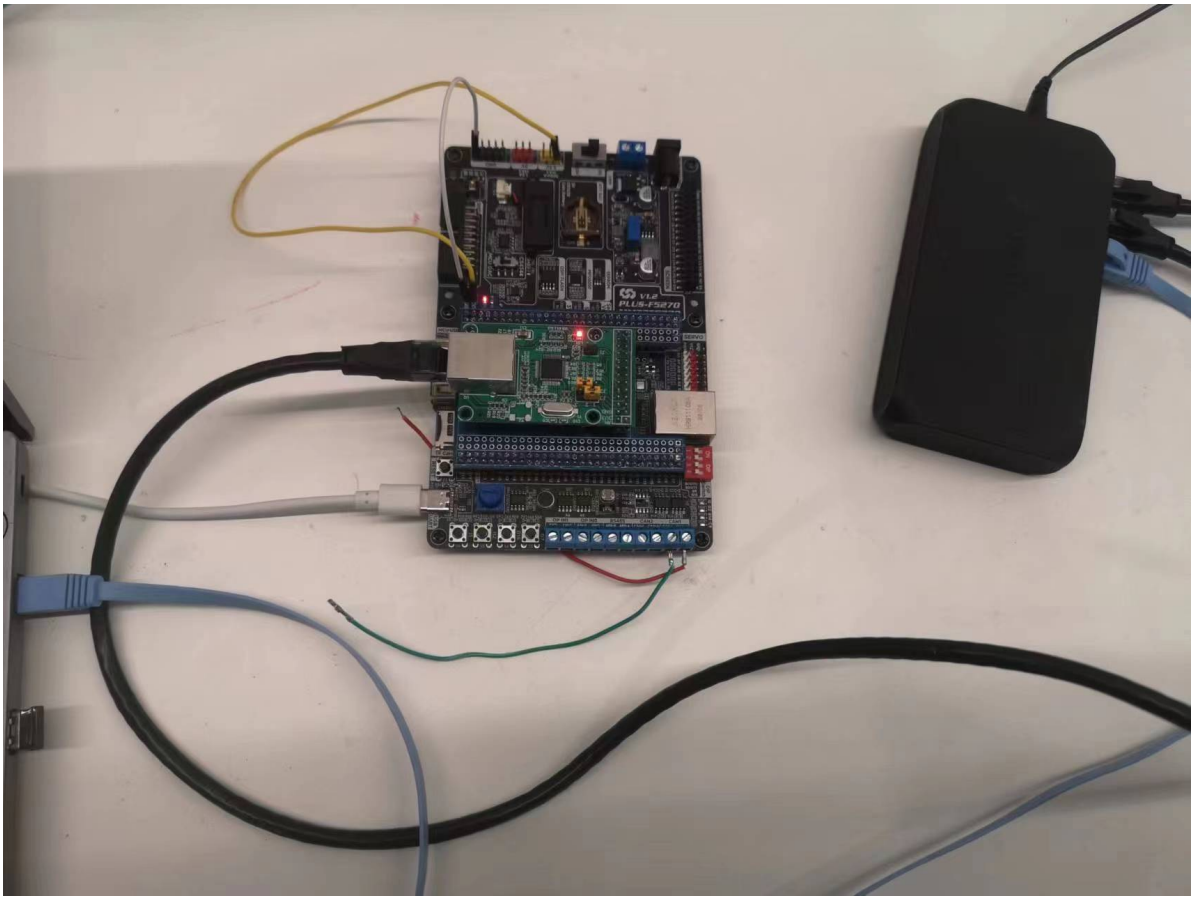

图4 以太网硬件连接

2. **安装并配置网络调试助手**

在官网(http://free.cmsoft.cn/reslink.php?id=205)下载网络调试助手NetAssist并安装。

打开电脑终端(CMD)，输入指令`ipconfig/all`，查看本机的以太网IP地址为`192.168.105.85`，如图5所示。

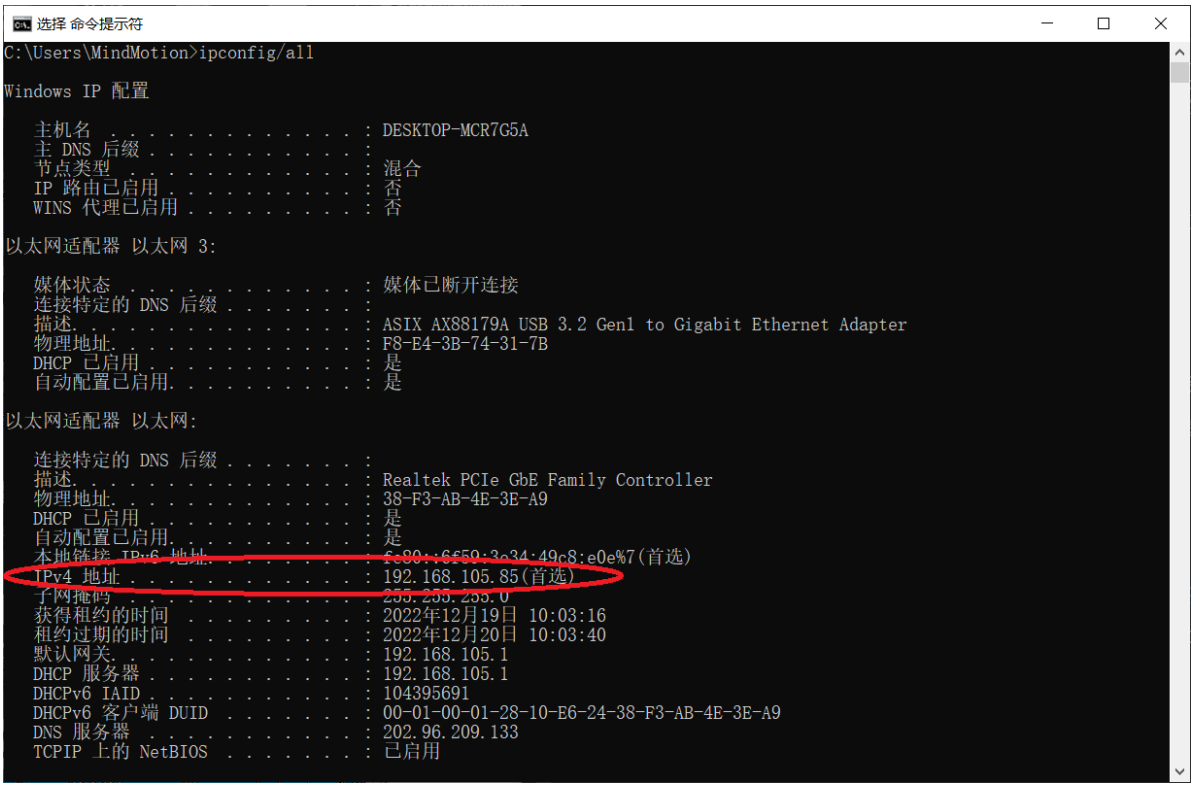

图5 本机的以太网IP地址

在网络调试助手左侧的协议类型选项中选择`TCP Server`，将enet_tcp_client样例的board_init.h文件中配置的目标服务器地址`192.168.105.85`填入网络调试助手的`本地主机地址`，将样例中配置的端口值`6800`写入网络调试助手的`本地主机端口`，样例中配置自身(客户机)IP地址为"192.168.105.98"。配置完成后，选择网络调试助手中的"打开"选项，开始运行服务器，等待客户端的命令。网络调试助手的配置如图6所示。

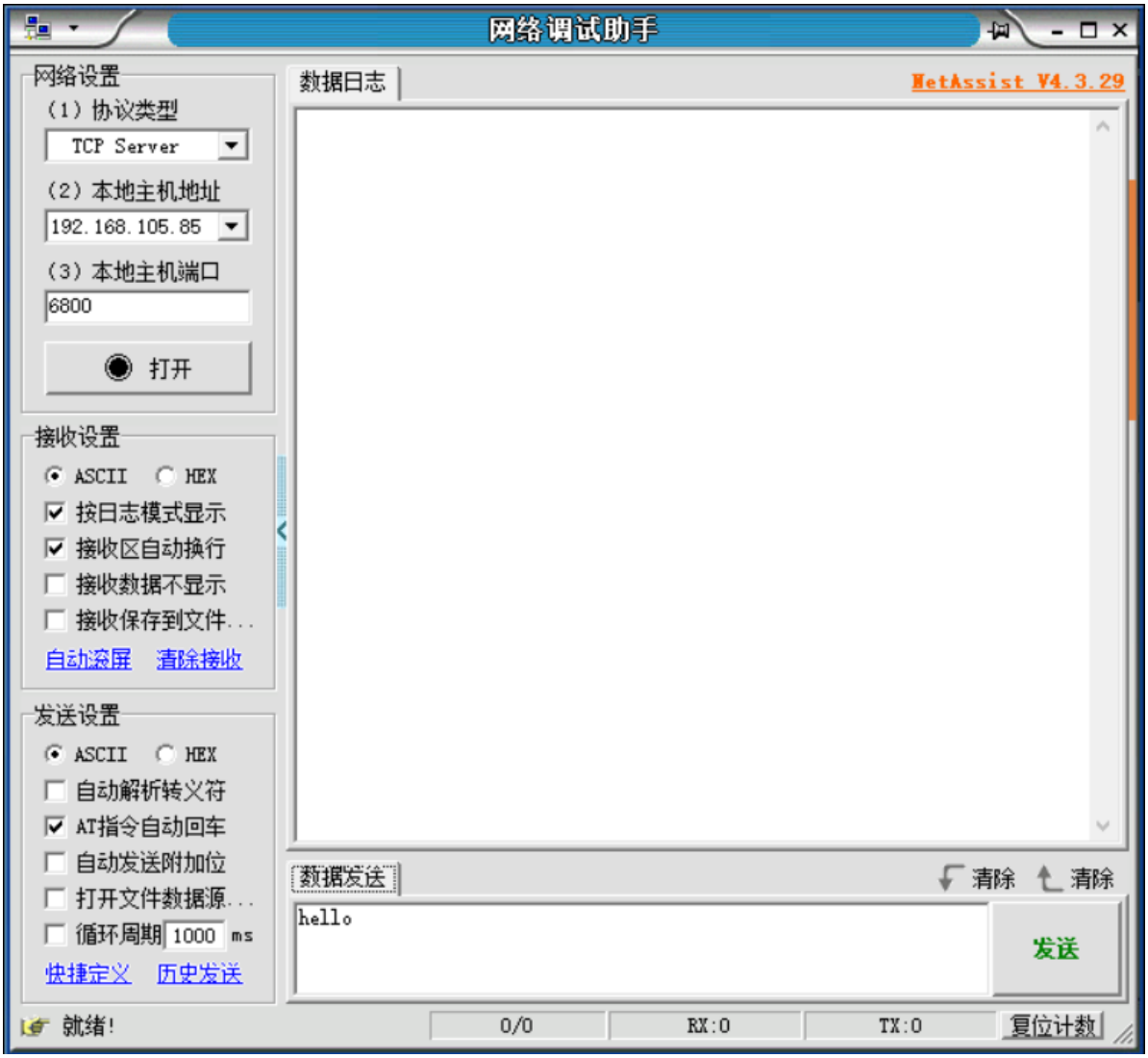

图6 调试所使用的网络调试助手

3. **下载并运行enet_tcp_client样例**

将MindSDK中的enet_tcp_client工程下载到开发板中并运行，若网络调试助手处于打开状态，则服务器可接收到客户端传来的数据字符串"New client connection."，使用网络调试助手进行数据发送，即客户端向传服务器发送数据，服务器接收到来着客户端的数据，向客户端发送相同的数据，并在网络调试助手上查看到客户机接收并发送的数据，该数据值与服务器发送数据相同。样例运行结果如图7所示。

图7 enet_tcp_client样例运行结果