V1.0 2007.2.6

RemoDAQ-8187 带 I/O 以太网控制器

模块用户手册

北京鼎升力创技术有限公司

目 录

1 系统概述 3
1.1 独立的数据采集和控制系统 3
1.2 特征 3
1.2.1 通过C语言编程可以实现灵活的控制功能 3
1.2.2 RS-232/485 通讯 3
1.2.3 通用通讯协议功能函数库 3
1.2.4 集成ROM和RAM的编程 4
1.2.5 集成实时时钟和看门狗定时器 4
1.2.6 集成的以太网口 4
1.3 RemoDAQ-8187 控制器的详细说明 4
1.3.1 系统 4
1.3.2 RS-232 接口（COM1） 5
1.3.3 RS-485 接口（COM2 和COM3） 5
1.3.4 RS-232/485 接口（COM4 和编程口） 5
1.3.5 电源 5
1.3.6 环境 5
2 安装指南 6
2.1 系统要求 6
2.1.1 主机配置： 6
2.2 硬件安装 6
2.2.1 跳线设置 6
2.3 系统接线和连接 7
2.3.1 电源线的连接 7
2.3.2 I/O接线 8
2.3.3 系统的连接 9
3 程序下载 11
3.1 系统硬件配置 11
3.2 软件配置 11
3.3 程序下载 12
4 网络功能函数库的使用 13
4.1 FTP 服务 15
4.2 HTTP服务器 16

4.3 发送Mail.....24
4.4 Modbus/TCP服务器.....27
4.5 MODBUS/TCP客户端.....30
4.6 MODBUS/RTU从属设备.....32
4.7 MODBUS/RTU 控制器.....34
4.8 TCP服务器和客户端.....35
4.9 UDP连接.....45
4.10 FTP客户端.....50
第 5 章 编程和函数库.....51
5.1 引言.....51
5.1.1 关于RemoDAQ-8187 控制器的编程细节.....51
5.1.2 Mini BIOS功能.....51
5.1.3 编译程序代码.....51
5.1.4 不同的内存模式对应不同的库类型.....52
5.1.5 限制.....52
5.1.6 看门狗定时器的编程.....52
5.2 函数库的类别.....53
5.2.1 类别A：系统功能（UTILITY*.LIB）.....53
5.2.2 类别B：I/O模块功能（HIO*.LIB）.....54
5.2.3 类别C：通讯功能：（A4_COMM*.LIB）.....54
5.2.4 类别D：MODBUS/RTU 功能（RTU*.LIB）.....54
5.2.5 类别E：MODBUS/TCP 功能（MBTCP*.LIB）.....55
5.2.6 类别F：网络套接字功能（SOCKET*.LIB）.....55
5.2.7 类别G：HTTP功能(CGI_LIB*.LIB).....56
5.3 函数库描述.....57
5.3.1 系统函数（UTILITY*.LIB）.....57
5.3.2 I/O 模块功能（HIO*.LIB）.....67
5.3.3 通信函数库（A4_COMM*.LIB）.....70
5.3.4 MODBUS/RTU 函数（RTU*.LIB）.....85
5.3.5 MODBUS/TCP 功能（MBTCP*.LIB）.....97
5.3.6 套接字函数（SOCKET*.LIB）.....108

1 系统概述

1.1 独立的数据采集和控制系统

RemoDAQ-8187 控制器是一个基于 X-86CPU 架构的以太网连接控制器。它不仅支持以太网而且还支持 4 个串行通讯口，使得 RemoDAQ-8187 特别适合与工业通信及控制应用。以太网控制功能包括：内建的 HTTP 服务，FTP 服务和 E-mail 功能。模块化的 I/O 设计为普通的应用设计提供了高度的灵活性。

RemoDAQ-8187 控制器提供了非常丰富的 Modbus 功能函数库，包括：Modbus/RTU(主/从)和 Modbus/TCP（服务器/客户端）功能函数库。

1.2 特征

RemoDAQ-8187 控制器系统由两个主要部分组成：主单元和 I/O 模块。主单元包括通讯接口，CPU 等等。I/O 模块包括 8 个数字 I/O 通道。

1.2.1 通过 C 语言编程可以实现灵活的控制功能

RemoDAQ-8187 控制器包含一个 80188CPU 和一个内建的 ROM-DOS 操作系统。用户可以象在办公室里使用一台 X86 结构的个人电脑一样操作它。

C 语言程序可以在 Borland C 3.0 里编辑和编译，然后下载到 RemoDAQ-8187 控制器里。

1.2.2 RS-232/485 通讯

RemoDAQ-8187 控制器有很强的通讯能力，有四个通讯端口，各个通讯口的模式如下：

COM1	RS-232（全信号）
COM2	RS-485
COM3	RS-485
COM4/编程	RS-485/RS-232

COM1 是一个全信号的 RS-232 端口，COM2 和 COM3 是 RS-485 专用端口，COM4 和编程口共享的，它可以通过跳线来选择具体的应用。

1.2.3 通用通讯协议功能函数库

RemoDAQ-8187 控制器的通讯协议是用户可定义的，并且提供了许多 Modbus/RTU 和 Modbus/TCP 协议的功能函数库可供用户使用。用户也可以执行他们自己的基于 ASCII 的通讯协议。

函数库包含以下协议：

- Modbus/RTU 控制功能，通过 RS-485 口远程控制 I/O 模块
- Modbus/RTU 从属功能,通过 RS-485 口连接 HMI/SCADA 软件
- Modbus/TCP 服务器功能,通过以太网口连接 HMI/SCADA 软件
- Modbus/TCP 客户端功能，通过以太网口连接具有以太网功能的远程 I/O 模块

1.2.4 集成 ROM 和 RAM 的编程

RemoDAQ-8187 控制器拥有集成的闪存和 SRAM 用来下载程序、运行系统和存储数据;提供 1MB 的文件系统(960KB 可供用户下载程序)。还提供了 640KB 的 SRAM 供应用程序和传输文件用。而且用户可以使用 384KB 的 SRAM 空间作为电池后备内存使用。

1.2.5 集成实时时钟和看门狗定时器

实时时钟用来确定时间，看门狗用来当系统出现错误时自动复位系统。这些功能使得 RemoDAQ-8187 控制器可以适应恶劣的工业现场环境，RemoDAQ-8187 更加稳定可靠。

1.2.6 集成的以太网口

RemoDAQ-8187 以太网口提供了以下功能：

- FTP 服务器和客户端功能
- WEB 服务功能
- 发送 MAIL 功能
- 通过套接字的 TCP 和 UDP 连接功能

1.3 RemoDAQ-8187 控制器的详细说明

1.3.1 系统

- CPU：16 位微控制器
- 内存：

  1.5MB 闪存

  - 256KB 系统盘（驱动器 C：只读）
  - 256KB 闪存（通过函数来访问）
  - 1024KB 文件系统,960KB 用户应用(驱动器 D：读/写)

  640KB SRAM

  - 电池备份可用 384KB（通过函数读取）
- 操作系统：ROM-DOS（MS-DOS 6.22 兼容）
- 实时时钟：有
- 看门狗定时：有
- RS-232 接口：COM1
- RS-485 接口：COM2，COM3
- RS-232/485 接口：编程口/COM4（跳线选择）
- 网络接口：10/100Base-T
- 板载 I/O 功能：

  4 通道数字量输入：

  干接点：

  逻辑电平 0：接地

逻辑电平 1：开路

湿接点：

逻辑电平 0：最大+2V

逻辑电平 1：4V~30V

4 通道数字量输出：

集电极开路输出 40V，200mA（最大负载）

1.3.2 RS-232 接口（COM1）

- 信号：Txd, Rxd, Rts, Cts, Dtr, Dsr, Dcd, RI, GND
- 模式：异步全双工，点对点
- 连接：DB-9 针
- 传输速度：最高达 115.2Kbps
- 最大传输距离：15.2m

1.3.3 RS-485 接口（COM2 和 COM3）

- 信号：DATA+，DATA-
- 模式：半双工，差分传输
- 连接：双绞线
- 传输速度：最大 115.2Kbps
- 最大传输距离：1220m

1.3.4 RS-232/485 接口（COM4 和编程口）

- RS-232/485 模式选择：通过跳线选择
- RS-232 模式（编程口）：全双工，点对点

  信号：TxD，RxD，GND
- RS-485 模式：半双工，差分信号

  信号：DATA+，DATA-
- 连接：双绞线
- 传输速度：最大 115.2Kbps
- 最大传输距离：

  RS-232：15.2m

  RS-485：1220m

1.3.5 电源

- 直流：10V~30V
- 电源功率：2.0W（典型值）

1.3.6 环境

- 工作环境：-10~70℃
- 储存温度：-25~85℃
- 湿度：5-95%，无结露
- 空气：无腐蚀性气体

2 安装指南

2.1 系统要求

在安装 RemoDAQ-8187 控制器之前，请确认您的系统符合以下要求：

2.1.1 主机配置：

1、 IBM 个人电脑，486CPU，或 PC 机(推荐奔腾级配置)
2、 Microsoft XP/2000/NT/98/95 或更高版本
3、 DOS 3.31 或更高
4、 DOS 版 Borland C 3.0
5、 最低 20MB 硬盘空间；VGA 彩色显示器
6、 2X 或更高速光盘驱动器；鼠标或者其它定位设备
7、 至少一个标准串口
8、 一块 RS-485 卡，或 RS-232 转 RS-485 转换器(例如 RemoDAQ-8520)，作为系统编程用通讯口
9、 以太网连接用 HUB 或交换机

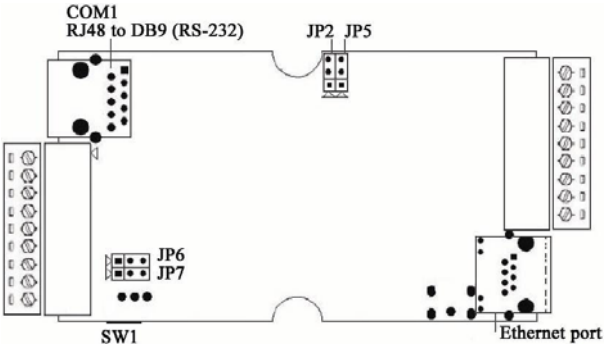
2.2 硬件安装

2.2.1 跳线设置

板上五个跳线的功能如下：

SW1：INIT 模式和正常工作模式
JP2：复位模式设置
JP5：电池后备 RAM 设置
JP6 和 JP7：COM4 口通讯模式选择

下图指出了各个跳线的位置：



复位模式设置

JP2 跳线允许用户来设置复位模式：

1、 看门狗复位模式：若看门狗定时器被允许，则需要至少每隔 1.6 秒清一下看门狗，否则系统会被复位。参考 5.3.1 了解怎样利用函数库设置看门狗定时器。

2、 NMI(非可屏蔽中断)复位模式：如果有非可屏蔽中断产生则系统复位。

3、 无复位模式：没有设置系统的复位模式。

出厂时的默认设置是“看门狗定时器复位模式”。

跳线设置见下图：

watchdog reset

NMI reset

non-reset

电池后备 RAM 设置：

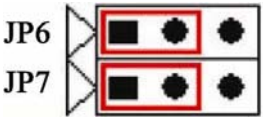
跳线 JP5 用来设置 SRAM 的电池后备功能是启用还是禁止。出厂默认设置是启用电池后备功能。跳线的设置见下图：

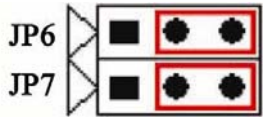
Enable

Disable

COM4 通讯模式选择设置：

COM4 的通讯模式由 JP6 和 JP7 来设置。出厂时 COM4 的默认设置是作为程序下载的编程口（RS-232 模式）。用户也可以设置 COM4 为 RS-485 模式。跳线的设置如下图：





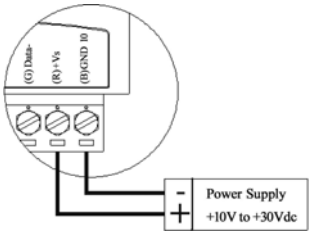
2.3 系统接线和连接

这一节提供了关于电源线、I/O 口，通讯口和编程口的基本连接方法。

2.3.1 电源线的连接

RemoDAQ-8187 控制器使用标准的工业直流 24V 电源，但只要在 10-30V 范围内的

直流电源都可为它提供电源。电源的纹波必须小于 200mV，电源线的线径至少在 2mm 以上。

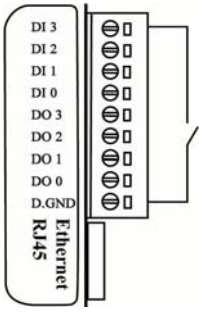


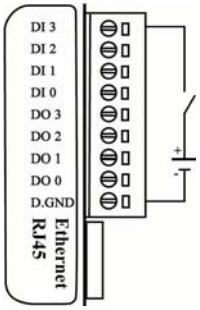
2.3.2 I/O 接线

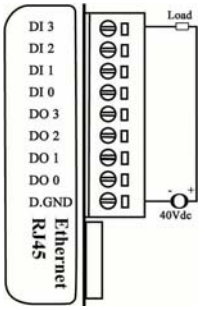
控制器使用一排接线端子来连接 I/O 模块和外部设备。以下是需要注意的问题：

1. 接线端子能接入的线径是 0.5mm 到 2.5mm
2. 尽量使用一条连续的导线
3. 使用尽量短的导线
4. 如果条件允许的话使用线夹子固定导线
5. 避免走线靠近高压电线
6. 若条件允许,尽量不要让输入线和输出线靠的太近

下图为外部信号连接 RemoDAQ-8187D 端子示意图。





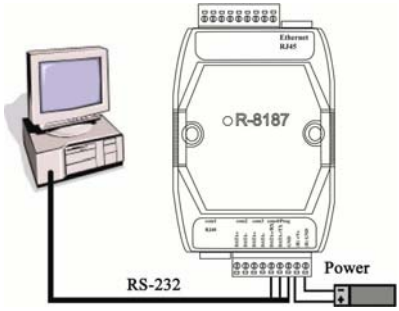


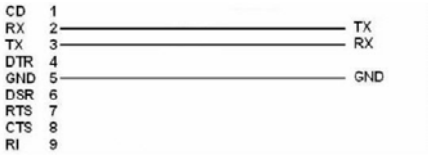
2.3.3 系统的连接

系统信息配置和下载的连接

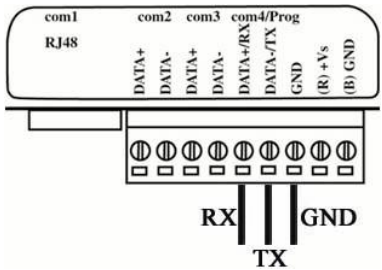
RemoDAQ-8187 有一个编程口,这个端口(Com4)允许用户通过上位机对 RemoDAQ-8187 进行编程、信息配置和调试。编程口通过 RS-232 接口的 TX，RX 和 GND 三根信号线与上位机连接。

导线的连接和针脚的定义见下图：



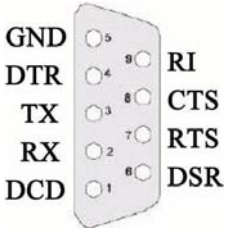


注意：鼎升力创公司提供了一条用来编程的导线，参考下图。



由于 RemoDAQ-8187 的 COM1 接口不是一个标准的 RS-232 接口，所以我们提供了一条 RJ48 转 DB9 的转接线，用以将控制器上的非标准 RS-232 接口转为标准的 RS-232 接口。

RemoDAQ-8187 的 DB9 针脚定义见下图：



RS-485 接口用于系统监视和集成

RemoDAQ-8187 控制器提供 RS-485 接口用于连接外部模块。Com2 和 Com3 是专用的 RS-485 接口,Com4 是一个 RS-232/485 可选的接口.它们都可用来提供 Modbus/RTU 主动和从动功能。

以太网连接

RemoDAQ-8187 控制器提供以太网接口(RJ-45 接头)用于网络连接。

3 程序下载

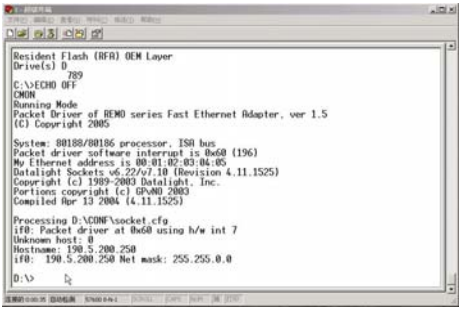
本章将讲述如何配置和下载应用程序。

3.1 系统硬件配置

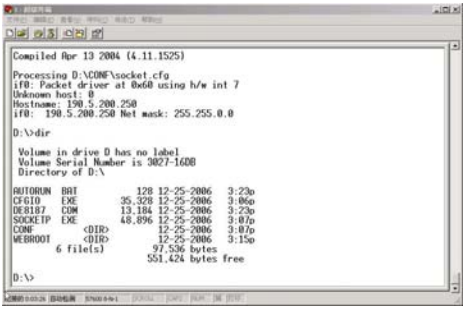
跳线 SW1 设置为 Initial 模式时重新启动控制器，则控制器将工作在配置和下载程序的模式下，这时您可以配置本控制器和下载应用程序到控制器中。当将跳线 SW1 设置为 Normal 模式时，重新启动控制器，控制器将自动执行在 autorun.bat 文件里配置好的程序。

3.2 软件配置

从“开始\程序\附件\通讯\超级终端”启动超级终端，新建一个连接，选择与 RemoDAQ-8187 连接的串口号，波特率设为“57600”，数据流控制设为“无”，其他为默认设置。打开 RemoDAQ-8187 电源，您将看到以下画面：



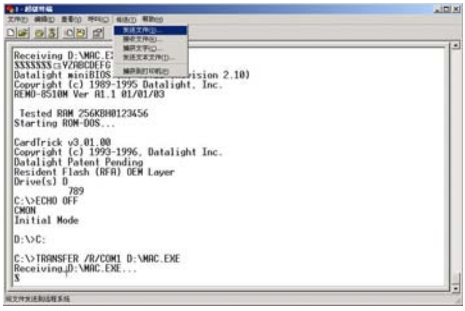
可输入“dir”查看 RemoDAQ-8187 存储器中的内容：

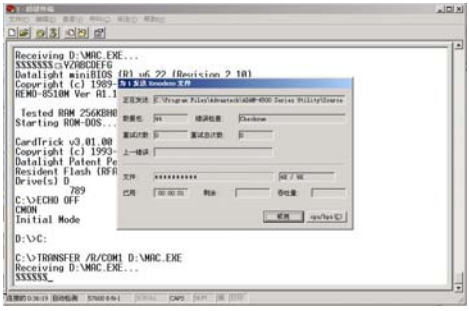


在这里可象在 DOS 系统里运行 DOS 命令一样，如“del”删除文件，“md”建立文件夹，“rd”删除文件夹等等。

3.3 程序下载

将 SW1 跳线跳在 INIT 模式，重新启动模块。切换到 C 盘根目录下，在超级终端中输入“Transfer/r/Com1 D:\文件名”在超级终端点传送下面的发送文件，如图所示：





传送完成后，可运行“DIR”命令来查看传送的文件。

传送完成后，将 SW1 跳线设置为“Normal”模式才可运行下载的程序。后面章节提到的下载程序均使用此方法来下载。

4 网络功能函数库的使用

RemoDAQ-8187 控制器的网络控制功能是非常丰富的。为了缩短您学习这些网络功能函数的时间，在这一章中我们将演示程序。这些示例程序也可在附带的光盘里面找到。在您测试网络功能之前，需要配置以下两个文件：

Socket.cfg：包含与配置信息有关命令的文本文件

Socket.upw：包含用户名和密码的文本文件

网络界面配置文件：Socket.cfg(默认的启动文件)；

Hosts (主机名称文件)

Socket.cfg 是一个包含配置信息命令的文本文件。空行和以#开头的行是与命令或配置信息无关的行。命令行用来设置 IP 地址，路由等参数。程序举例：

IP address 192.6.1.1

设置本控制器的 IP 地址为：192.6.1.1

Interface pdr if0 dix 1500 5

使用分组报文，接口名称为：‘if0’,MTU=1500，接收缓存=5

Route add default if0 router

将 router 设置为网关

TCP mss 1460

TCP 最大分组大小为：1460

TCP window 2920

TCP 最大窗口数量为 2920

Hosts 是一个可选文件，包括 IP 地址到主机名称的对应

Hosts 示例文件：

```
192.6.1.1 demo
192.6.1.2 router
192.6.1.3 server
```

Socket.cfg 示例：

下面的示例指定了静态 IP 地址，一个报文分组接口，一个默认的路由，TCP 最大分组数和一个窗口

Socket.cfg：

```
#Sample configuration file
ip address 192.6.1.1
interface pdr if0 dix 1500 5
route add default if0 192.6.1.2
tcp mss 1460
tcp window 2920
```

“Socket.upw”格式

这个文件用来设置 HTTP 和 FTP 服务器的用户权限,每一行都包含了一个用户的信息，若行是以‘#’开头的，则表示它是一个注释行。每一行都包括以下四个部分：

<用户名><密码><工作目录><权限>[#注释]

- 用户名：用户的名称。如果是*，则当客户机没有指定一个用户名时将以此权限登陆
- 密码：用户的密码。如果是‘*’，则不需要输入密码
- 工作目录：用户将只允许访问这个目录和它的子目录，如果是‘/’，用户将可访问整个系统。HTTP_DIR 可以用‘\’来代替，如果一个相关的路径已经被指定，那么 HTTP_DIR 也将被附加指定。
- 权限：若一个用户在 FTP 和 HTTP 都被授予一定的权限，那么 FTP 权限代码应该写在前面，否则将被忽略。权限代码如下：

FTP 权限：

d	改变目录
c	创建/删除目录
w	写文件
r	读文件

HTTP 权限：

e	复制文件
p	粘贴文件
g	使用 CGI
m	使用远程控制

字段之间需要用一个空格来分隔，如果字段之间没有空格，则这个字段将被忽略。注释跟在最后一个字段（权限）后面。

下面是两个配置文件的示例：

SOCKET.CFG：

```
#Packet driver settings
ip address 192.168.1.4
interface pdr if0 dix 1500 10 0x60
#The following will cause SOCKETS to display IP status ip address

#The following lines set TCP parameters
ip ttl 64
tcp mss 1460
tcp window 2920
```

Socket.upw：

```
Su su / drwcepgm        #su can do everything on whole system.
* * \guest rg           #everyone can read(FTP)and get (HTTP)
                        #from %HTTP_DIR%\guest

test1 test1 \ drep   #test1 can change directories and read files(FTP)
                     #test1 get and post files (HTTP) in %HTTP_DIR%\

ftp1 ftp1 \guest rd   #ftp1 can read files and change directories (FTP)
                      #in %HTTP_DIR%\guest ftp1 has no HTTP rights

http1 http1 / epgm   #http1 can get and post files,use CGI,and use remote console.
                     #http1 has no FTP rights

user1 user1 \user\user1 rdcw   #user1 has full FTP access rights to
                        # the directory %HTTP_DIR%\user\user1
                        #user1 has no HTTP rights
```

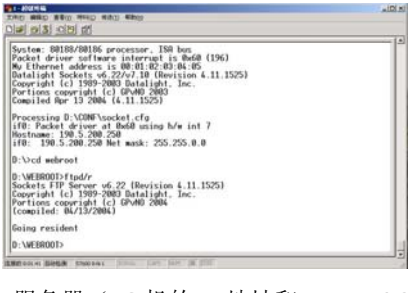
4.1 FTP 服务

测试程序：FTPD.EXE

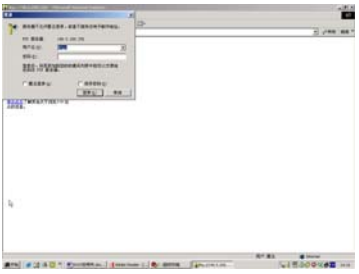
系统 配置：

- 在 RemoDAQ-8187 控制器上运行 FTP 服务程序
- 在 PC 机上运行 FTP 客户端程序

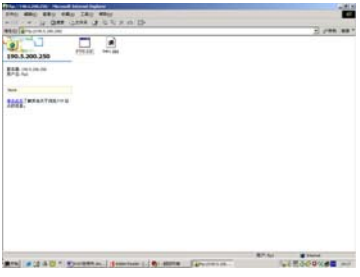
1. 将 FTPD.EXE 下载到 RemoDAQ-8187 的驱动器 D 上的“Webroot”目录下(跳线跳在 Initial 模式)。
2. 将跳线跳回 Normal 模式，并重新启动模块。输入“cd webroot”进入“webroot”目录，输入“ftpd /r”以常驻模式运行 FTPD.EXE。



3. 在 PC 机上连接 FTP 服务器（PC 机的 IP 地址和 RemoDAQ-8187 控制器 IP 地址必须在同一个域里。
4. 通过输入用户名和密码登陆 FTP 服务器（用户名和密码在 Socket.upw 文件里）



5. 查看在“webroot”目录下面的文件是否正确。



4.2 HTTP 服务器

示例 1：

示例程序：Httpdemo.exe（无 CGI 功能）

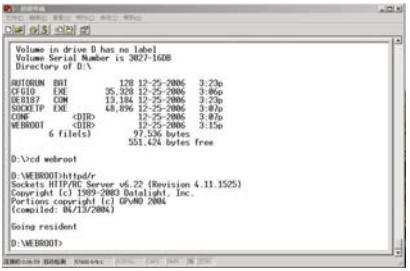
源文件：Httpdemo.c 在光盘的 RemoDAQ-8187\Source\Example\Httpex 目录下。

演示文件：Httpd.exe

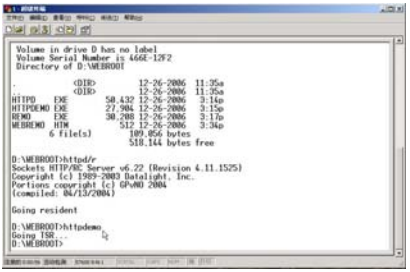
系统配置：

- 在 RemoDAQ-8187 上运行 HTTP 服务器程序
- 在 PC 上使用 WEB 浏览器来连接 HTTP 服务器

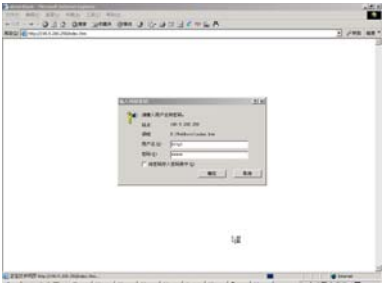
1. 下载 HTTPD.EXE (\RemoDAQ-8187\Source\Drive_D\Extension_files)到 RemoDAQ-8187 上的 D 驱动器的 “webroot”目录下(跳线在 Initial 模式)。
2. 将 RemoDAQ-8187\SOURCE\example\httpEx 目录下 Httpdemo.prj 文件编译为 Httpdemo.exe 文件，将它下载到 RemoDAQ-8187 驱动器 D 上“webroot”的目录下。
3. 将跳线设置为 Normal 模式并且重新启动。输入“cd webroot”进入“webroot”目录，输入“httpd /r”以常驻内存方式来运行程序。



4. 输入“httpdemo”来运行 HTTPDEMO.EXE。

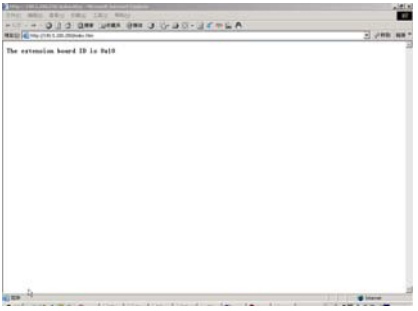


5. 在PC机上运行IE浏览器，输入 http://190.5.200.250/index.htm，通过输入用户名和密码来登陆系统。

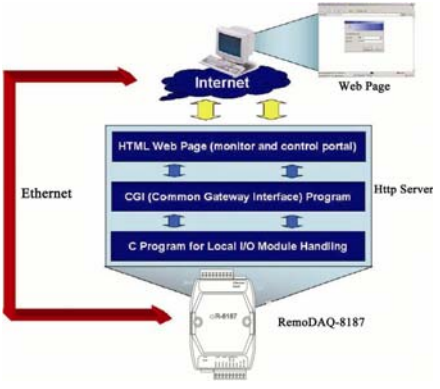


注：190.5.200.250 是 RemoDAQ-8187 控制器的默认 IP 地址，通过 Socket.cfg 获得当前的 IP 地址，也可通过 Socket.upw 获得用户名和密码。

6. RemoDAQ-8187 控制器的内建 I/O 模块的 ID 如下图



下图是 HTTP 服务器功能的软件结构。HTTP 服务程序已经集成在 RemoDAQ-8187 以太网控制器里，当用户打开 IE 浏览器通过 Internet 或 Intranet 获取 RemoDAQ-8187 数据的时候，它将使用以有的网页提供一个监视和控制的入口。所有来自网页的命令必须通过 CGI 程序来连接到可以实时读写 I/O 通道的 C 语言程序上。



进行网络监视和控制的时候通常有三步：

1. 注册：在网页上注册一个事先确定好的名称
2. 用户登陆和调用控制程序：注册之后，用户就可以通过登陆服务器来调用本地控制程序
3. 通过本地控制程序来控制本地 I/O

Httpdemo.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>

#include "8187drv.h"
```

```
#include "CGI_Lib.h"

FILE *fp;
int number = 0;
int count = 1;

void ReplaceStr(char *ptr_str1, char *ptr_str2, int len_str);

void main(void)
 {
    char * homepage_name = "index.htm";

       if(!Http_Server_Reg(homepage_name))
            return;
       printf("Program exiting...");
       HttpDeRegister("index.htm");
 }

int far Callback(HTTP_PARAMS far* psParams)       //implement your program in this function
 {
    static char *ptr_OO = 0;
    char *tmpStr = 0;
    static char Htm_Content[]="HTTP/1.0 200 OK\r\n"
      //content of html page, content=1
      "Content-type:text/html\r\n\r\n"
      //means refreshes every 1 second
     "<html><meta http-equiv=""Refresh""content=1>"
      "<b>The extension board ID is OOOOOOO</b><p>"
      "</html>";

       number++;
       printf("Refresh %d times...\n", number);

       if (!ptr_OO)
            ptr_OO = strstr(Htm_Content, "OOO");

       sprintf(tmpStr, "0x%X", Get_BoardID());
       ReplaceStr(ptr_OO, tmpStr, 7);

   HttpSendData(psParams->iHandle,tm_Content, trlens
     (Htm_Content));
return RET_DONE;
```

```
}

    void ReplaceStr(char *ptr_str1, char *ptr_str2, int len_str)
    //replace string
{
     int i;
     for(i=0; i<len_str; i++)
          ptr_str1[i] = 32;

     for(i=0; i<strlen(ptr_str2); i++)
          ptr_str1[i] = ptr_str2[i];
}
```

示例 2

示例程序：Remo.exe（具有 CGI 功能）

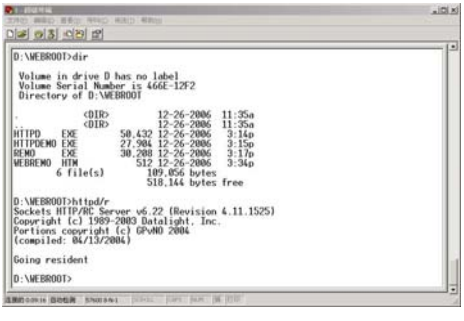
源文件：Remo.c 和 Webremo.htm(RemoDAQ-8187\Source\Example\httpEx 目录下)

程序：Httpd.exe

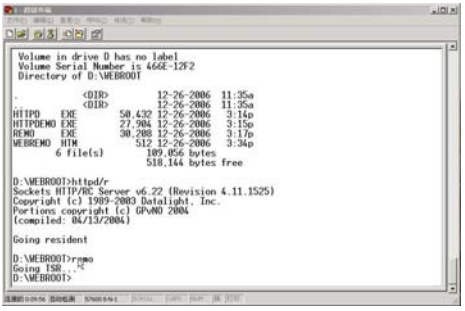
系统配置：

- 在 RemoDAQ-8187 控制器上运行 HTTP 服务器程序
- 在 PC 机上使用网页浏览器来连接到 HTTP 服务器

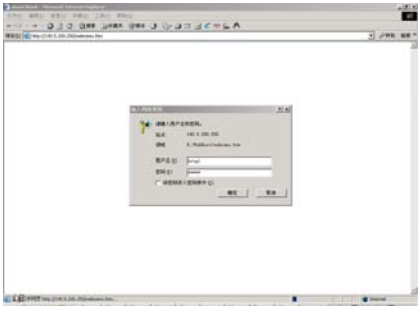
1. 下载 HTTPD.EXE(在 RemoDAQ-8187\Source\Drive_D\Extension_files)到驱动器 D 的“webroot”目录下(跳线设置为 Initial 模式)
2. Remo.prj 文件编译为 Remo.exe(在 RemoDAQ-8187\Source\ Example\ httpEx), Remo.exe 下载到驱动器 D 的“webroot”目录下
3. 下载 Webremo.HTM 到驱动器 D“webroot”目录下。
4. 跳线设置 Normal 模式并重新启动。输入“cd webroot”进入“webroot”目录，输入“httpd /r”以常驻内存模式运行 Httpd.exe



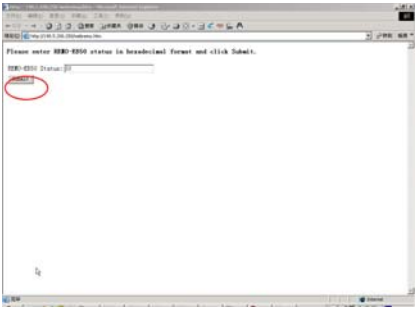
5. 输入“Remo”来运行 Remo.exe



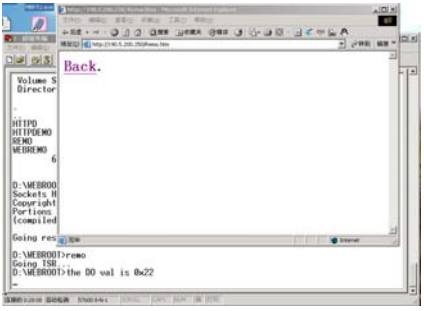
6. 在PC机上运行IE，输入 http://190.5.200.250/webremo.htm (注：190.5.200.250 是 RemoDAQ-8187 默认的 IP 地址，Socket.cfg 获得模块的当前 IP 地址，可参考 Socket.upw 获得用户名和密码。)



7. 可通过 IE 设置 RemoDAQ-8187 集成 I/O 模块状态。



8. 可以通过超级终端查看设置值



Remo.c

```
#include <stdio.h>
#include <io.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>

#include "8187drv.h"
#include "CGI_Lib.h"

extern unsigned _stklen = 3000;
extern unsigned _heaplen = 2000;

int far Callback(HTTP_PARAMS far* psParams);
int returnVal(char *ptr_name, char *ptr_end);
int count = 1;

void main(void)
{
      char * homepage_name = "REMO.htm";

      if(!Http_Server_Reg(homepage_name))
            return;

      printf("Program exiting\n");
      HttpDeRegister("REMO.htm");
}

int far Callback(HTTP_PARAMS far* psParams)
//implement your program in this function
   {
         char buf[200],*p,*ptr_val,*ppass;
```

```
        int iQueryLen;
        char Re_Htm_Content[400];
        char *ptr_Re = Re_Htm_Content;
        int numberbytes;
        int DoVal, DIVal;

    *buf = 0;

    iQueryLen = _fstrlen(psParams->szQuery);
    if (iQueryLen)
        _fmemcpy (buf,psParams->szQuery, iQueryLen);

numberbytes=HttpGetData(psParams->iHandle,buf +iQueryLen,
200 - iQueryLen);

  if (numberbytes < 0)
  {
      if (numberbytes == (-WOULDBLK))
          return RET_OK;
      else
              printf("wrong input value\n");
  }

    iQueryLen += numberbytes;

    ptr_Re += sprintf(ptr_Re, "HTTP/1.0 200OK\r\n Content-type: text/html\
r\n\r\n<html><h1>");
    if (strncmp(buf,"DOValues=", 9) == 0) {
        ptr_val = buf + 9;
        if ((p = strchr(ptr_val,'&')) == NULL)
            printf("Please click Submit button..\n");

      printf("the DO val is 0x%x\n", returnVal(ptr_val, p));
      SetDO(EB50_ID, AllChannels, 0, returnVal(ptr_val, p));
    }
  ptr_Re += sprintf(ptr_Re, "<P><P><A HREF=\"/Web Remo.
htm\">Back</A>.</html>\n");
  Httpsenddata(psparams->ihandle,re_htm_content, ptr_re–re_htm_content);
    return RET_DONE;
}

int returnVal(char *ptr_name, char *ptr_end)
{
    int r_Val, buf_idx;
    char buf_val[10];
```

```
        memset(buf_val, 0, 10);
        for(buf_idx=0; buf_idx<10; buf_idx++)
        {
            if(ptr_name == ptr_end)
                break;
            buf_val[buf_idx] = ptr_name[buf_idx];
        }

        sscanf(buf_val, "%X", &r_Val);
        return r_Val;
    }
```

Webremo.htm

```
    <html>
    <head>
    </head>

    <body>
        <b>
        <p><p><p><p>
    Please enter REMO-EB50 status in hexadecimal format and click Submit.
        </b>

        <form action="Remo.htm" method=post name="login1">

        <table>
        <tr>
        <td>REMO-EB50 Status:</td>
        <td align=right><input name="DOValues" type=text
    size=30 maxlength=50></td>
        </tr>
        <tr>
        <td>
        <input name="submit" type=submit value="Submit">
        </td>
        </tr>
        </table>

    </body>
    </html>
```

4.3 发送 Mail

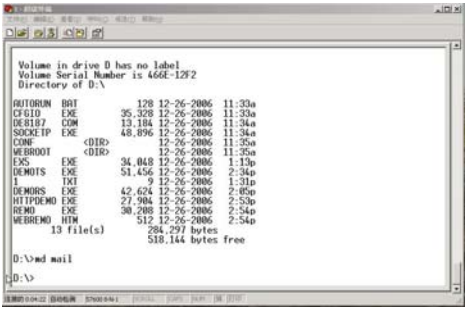
示例程序：Amail.exe；Mall.txt

源文件：Alarmmall.c 在 RemoDAQ-8187/Source\Example\mail 目录下

程序：Sendmall.exe；Makemall.exe

RemoDAQ-8187 配置：

1. md 建立一个新目录：md mall(跳线为 Initial 模式)



2. 在 PC 机上建立一个“Mail.txt”文件，编辑并保存。此文件即是发送 e-mail 的文件。
3. Mail.txt 下载到驱动器 D 的“Mail”目录下。
4. Makemall.exe 和 Sendmall.exe(在 RemoDAQ-8187\Source\Drive_D\Extension_files 目录下)下载到驱动器 D 的“Mail”目录下。
5. Amail.prj 文件编译成 Amail.exe 文件(在 RemoDAQ-8187\Source\Example\mail 目录下)，并下载 Amail.exe 文件到驱动器 D 的“Webroot”目录下。

   将跳线设置为 Normal 模式并重新启动。输入“cd mail”进入“Mail”目录。输入“amail”运行 Amail.exe。这个程序将执行 Makemall.exe 程序创建 e-mail 的内容，通过输入 DO 的输出值触发程序发送 e-mail。
6. 执行 Sendmail.exe 发送刚才创建的 e-mail。
7. 检查信箱查看收到的 email 是否正确。

注：RemoDAQ-8187 的 IP 地址要和 Mail 服务器的 IP 地址在一个域中，这样将使您能够从 RemoDAQ-8187 发送 email。如果您的 mail 服务器的 IP 地址和您的控制器不在一个域里面，mail 服务器将验证 email 的发送 IP 地址并停止提供服务。

ALARMMAIL.C

```
#include <stdio.h>
#include <process.h>
#include <errno.h>
#include "8187drv.h"

int SendAlarmMail(void);
int MakeAlarmMail(void);
int count = 1;
void main(void)
```

```
{
    unsigned long div, dov;
    unsigned int tmpcnt;

    if(!MakeAlarmMail())
    {
        printf("make mail fail..");
        return;
    }

    while(1)
    {
        printf("Please input digital output values: ");
        scanf("%X", &dov);

        if(dov == 0x33)
            return;

        SetDO(EB50_ID, AllChannels, 0, dov);
        printf("DO value 0x%X, press any key to continue\n", dov); getch();
        GetDIO(EB50_ID, AllChannels, 0, &div);

        if(div == 0x000f)
        {
            if(!SendAlarmMail())
            {
                printf("send mail error..");
                return;
            }
        }
    }

}

int MakeAlarmMail(void)
{
    char * arg_To = "-t567@123.com";
    char * arg_From = "-f345@hotmail.com";
    char * arg_subject = "-sREMO8187TCP";
    char * arg_MailContent = "-bmail.txt";
    char * arg_O_mail = "-omail.dat";

    printf("Making Mail..\n");
    if(spawnlp(P_WAIT,
                "d:\\mail\\makemail.exe",
                "d:\\mail\\makemail.exe",
                arg_To,
```

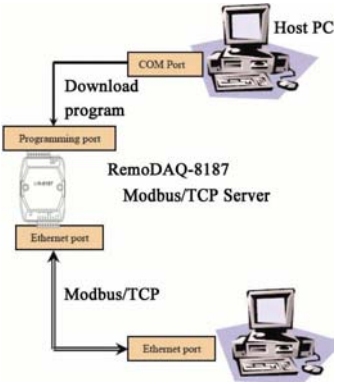
```
                        arg_From,
                        arg_subject,
                        arg_MailContent,
                        arg_O_mail,
                        NULL)==-1)
        {
            return 0;
        }
        return 1;
    }
    int SendAlarmMail(void)
    {
        char * arg1 = "smtp.123.com";
        char * arg2 = "mail.dat";
        printf("send Alarm mail prepare..\n");
  if(spawnlp(P_WAIT,"d:\\mail\\sendmail.exe","d:\\mail\\sendmail.exe",arg1,arg2,NUL
L)==-1)
        {
            return 0;
        }
        return 1;
    }
```

4.4 Modbus/TCP 服务器

示例程序：Demots.exe

源文件：Demots.c 在 RemoDAQ-8187\Source\ Example\DemoModbus 目录下

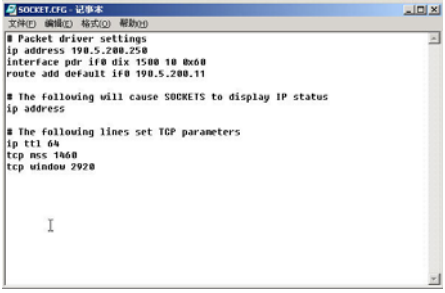
硬件配置：见下图



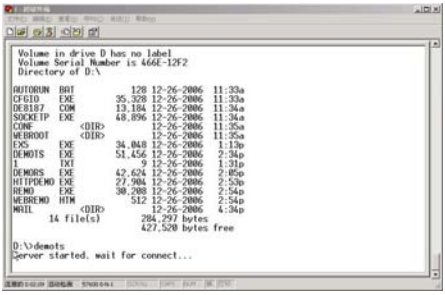
在这个例子中使用 RemoDAQ-8187 控制器作为 Modbus/TCP 服务器，一台在同一个网络里的 PC 机作为 Modbus/TCP 客户端。RemoDAQ-8187 控制器往 40001 地址里

写板载 DIO 值，Modbus/TCP 客户端应用程序可以从这个地址读这些值。(共享内存的地址 0 代表 Modbus 的地址 40001)

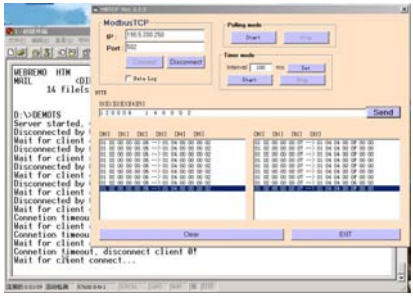
1. 在 Socket.cfg 文件里配置 RemoDAQ-8187 的 IP 地址，并将 Socket.cfg 文件下载到 RemoDAQ-8187 控制器里。



2. 将 Demots.prj(在 RemoDAQ-8187\Source\Example\DemoModbus 目录下)编译为 Demotx.exe 文件，并下载到 RemoDAQ-8187 控制器里的驱动器 D 目录下。(在 RemoDAQ-8187 应该为 Initial 模式)
3. 将跳线设为 Normal 模式并重新启动。在 D 盘根目录下运行 Demots.exe，这样将在 RemoDAQ-8187 控制器上运行 Modbus/TCP 服务器程序



4. 您可以在 PC 机上通过相应的 Modbus/TCP 程序来获取 DIO 状态。



DEMOTS.C

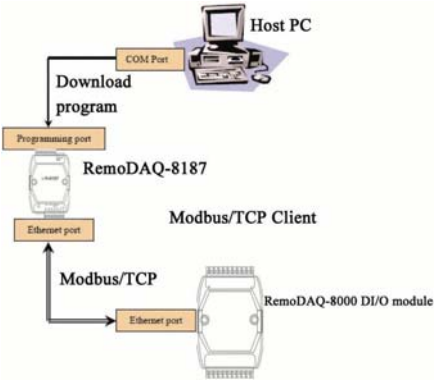
```
#include "mod.h"
#include "8187drv.h"

#define DATASIZE 250
#define sizeofShareMem        4000
int count=0;
int main(void)
{
   Socket Sock_8187;
   int err_code;
   unsigned int Share_Mem[sizeofShareMem];
   unsigned int tmpcnt=0;
   int tmpidx;
   unsigned long div;
    memset(Share_Mem, 0, sizeof(Share_Mem));
   if((err_code=REMOTCP_ModServer_Create(502, 5000, 7,
            (unsigned char *)Share_Mem, sizeof(Share_Mem)))!=0)
    //first step
    {
         printf("error code is %d\n", err_code);
    }
    Timer_Init();
    tmpidx = Timer_Set(1000);
    printf("Server started, wait for connect...\n");
    while(1)
    {
     RemoTCP_ModServer_Update();       //second step: return 0 NO packet, return 1
has packet
         if(tmArriveCnt[tmpidx])
         {
               Timer_Reset(tmpidx);
               disable();
               GetDIO(EB50_ID, AllChannels, 0, &div);
               Share_Mem[0] = div;
               //write DIO status to modbus address 40001
               enable();
         }
   }
RemoTCP_ModServer_Release();
return 0;
}
```

4.5 MODBUS/TCP 客户端

示例程序：Demotc.exe

源文件：Demotc.c (在 RemoDAQ-8187\Source\Example\DemoModbus 目录下)

硬件配置：见下图。



这个例子将 RemoDAQ-8187 作为客户端使用。

1. 将 Demotc.prj 编译为 Demootc.exe 下载到 RemoDAQ-8187 控制器中。
2. RemoDAQ-8187 运行 Demotc.exe 程序，RemoDAQ-8187 将作为客户机使用。

DEMOTC.C

```
#include "mod.h"

#define Server_Port 502
#define MAXDATASIZE 100

int main(int argc, char *argv[])
{
    char * ServerIP;
    SOCKET SO_8187;
    unsigned char HostData[MAXDATASIZE];
    int DataByteCount = 0;
    int tmp;
    unsigned int tmpcnt=0, tmpcnt1=0;
    int errcode;

    memset(HostData, MAXDATASIZE, 0);

    if(argc==2)
    {
        ServerIP = argv[1];
```

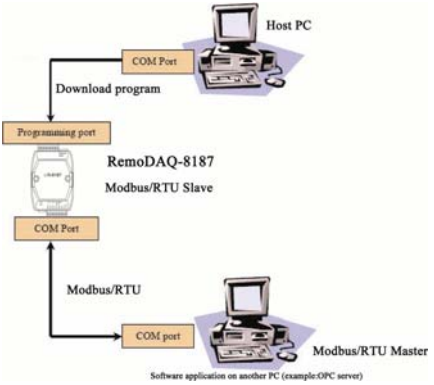
```
        }
        else
        {
            printf("Please input Server IP.\n");
            return 0;
        }

        if(REMOTCP_Connect(&SO_8187, ServerIP, Server_Port)<=0)
        {
             perror("REMOTCP_Connect()\n");
             REMOTCP_Disconnect(&SO_8187);
             return 0;
        }

            printf("Starting to send..\n");
            while(1)
            {
              //Query REMO-7051 Server
          if((errcode=REMOTCP_ReadCoilStatus(&SO_8187, 50, 0x01, 0x01, 0x01,
&DataByteCount, HostData))<=0)
        {
             if(errcode==TCPTimeOut_Err)
                  perror("Time Out.\n");
             else
             printf("Error: Error Code is %d\n", errcode);
           REMOTCP_Disconnect(&SO_8187);
            return 0;
        }
        else
        {
                  printf("REMO-7051 Channel 0 Status: ");
                  for(tmp=0; tmp<DataByteCount; tmp++)
             {
                 printf("%2X", HostData[tmp]&0x01);
                  }
             printf("\n");
             }

             for(tmpcnt=0; tmpcnt<50000; tmpcnt++)        //delay
             {for(tmpcnt1=0; tmpcnt1<4; tmpcnt1++){}}
        }
        return 1;
    }
```

4.6 MODBUS/RTU 从属设备

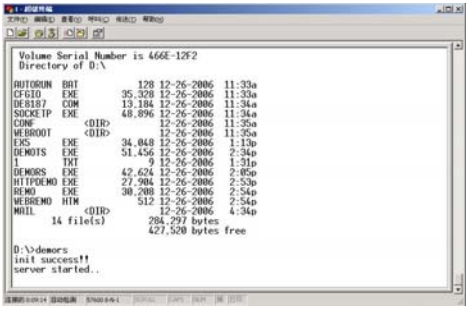
示例程序：Demors.exe

源文件：Demors.c

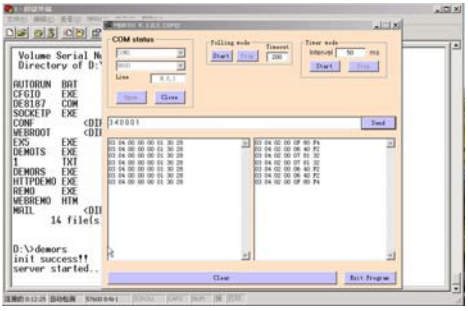
硬件配置：见下图



1. 将 Demors.prj 编译为 Demors.exe 文件，下载到 RemoDAQ-8187 控制器中。
2. 运行 Demors.exe，RemoDAQ-8187 控制器将写 DIO 值到地址 40001。



3. 运行相应的 PC 程序就可以看到 IO 值的变化。



DEMORS.C

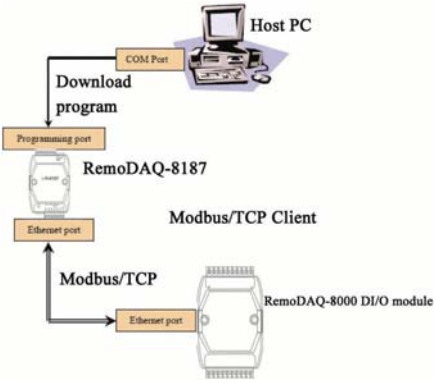
```
#include <stdio.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include "8187drv.h"
#include "RTU.h"
#define MAXDATASIZE 100
#define sizeofShareMem        10
int count;
void main()
{
   unsigned int Share_Mem[sizeofShareMem];
   char cCh;
   char LSR_State;
   unsigned int tmpcnt, tmpcnt1;
   unsigned long div;

   if(Modbus_Com_Init(Com2,Slave,(unsigned long)9600,No_parity,Data8,stop1)!=0
        {
          printf("error\n");
          return;
        }
        printf("init success!!\n");
   RemoRTU_Modserver_create(3,(unsigned char *)share_mem, sizeof(share_mem));
   //   {
   //          printf("err code is %d\n", Error_Code());
   //          return;
   //   }
   printf("server started..\n");
   while(1)
   {
        disable();
        GetDIO(EB50_ID, AllChannels, 0, &div);
        Share_Mem[0] = div;
        //write DIO status to modbus address 40001
        enable();
        for(tmpcnt=0; tmpcnt<50000; tmpcnt++)        //delay
        {for(tmpcnt1=0; tmpcnt1<8; tmpcnt1++){}}
   }
}
```

4.7 MODBUS/RTU 控制器

示例程序：Demorc.exe

源文件：Demorc.c

硬件配置：见下图



1. 将 Demorc.prj 编译为 Demorc.exe，下载到 RemoDAQ-8187 控制器。
2. 运行 Demorc.exe，建立连接。

DEMORC.C

```
#include <stdio.h>
#include <dos.h>
#include <time.h>
#include "RTU.h"

#define MAXDATASIZE 100

void main()
{
   unsigned char HostData[MAXDATASIZE];
   int cnt=0;
   unsigned int tmpcnt=0, tmpcnt1=0;

   if(Modbus_com_init(Com2,master,(unsigned long)9600,no_parity,data8,stop1)!=0
   {
      printf("error\n");
```

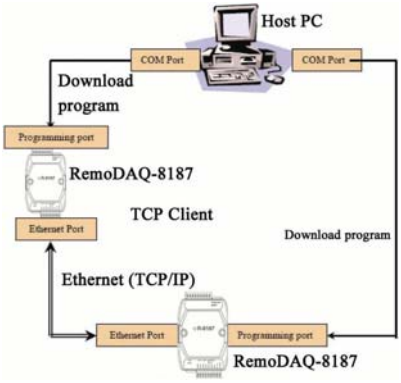
```
        return;
    }

    printf("init success!!\n");

    while(1)
    {

        cnt++;
        if(cnt%2==0)
          {
            HostData[1]=0x0f;
            HostData[0]=0xff;
          }
          else
          {
            HostData[1]=0x00;
            HostData[0]=0x00;
          }
          if(cnt==10)
            cnt = 0;

          //Set 8056S status
          if(!RemoRTU_ForceMultiCoils(Com2,0x03,0x11,0x0c,0x02,HostData))
          {
            printf("err code is %d\n", Error_Code());
            printf("fail send..");
          }
          else
            printf("Success!!\n");

    for(tmpcnt=0; tmpcnt<50000; tmpcnt++)      //delay
    {for(tmpcnt1=0; tmpcnt1<4; tmpcnt1++){}}
  }
}
```

4.8 TCP 服务器和客户端

示例程序：Tserver.exe 和 Tclient.exe

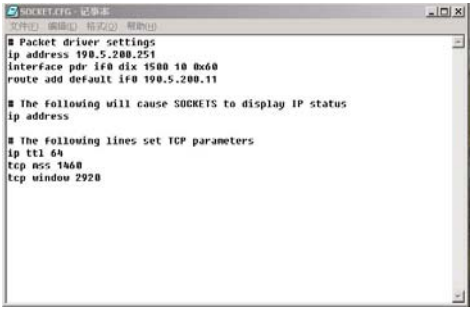
源文件：TCP_Server.c 和 TCP_Client.c

硬件配置：见下图

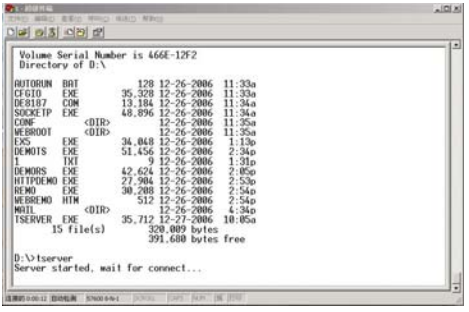


1. 参考上面章节设置 RemoDAQ-8187 TCP 服务器的 IP 地址

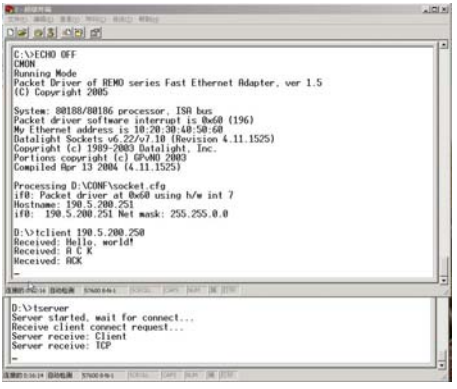
2. 参考上面章节设置 RemoDAQ-8187 TCP 客户机的 IP 地址



3. 将 Tserver.prj 编译为 Tserver.exe 并下载到做为 TCP 客户端 RemoDAQ-8187 中
4. 将 Tclient.prj 编译为 Tclient.exe 并下载到做为 TCP 客户端 RemoDAQ-8187 中
5. 在作为 TCP 服务器的 RemoDAQ-8187 上运行 Tserver.exe 程序



6. 在作为 TCP 客户端的 RemoDAQ-8187 上运行 Tclient.exe 程序



TCP_SERVER.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Errno errno

#define FALSE 0
```

```
#define TRUE 1
#define Host_Port 9510
#define Max_Conn 40
#define MAXDATASIZE 100

SOCKET remoteSocket[Max_Conn];
int WaitSocketCount[Max_Conn];
int socketTotal = 0;
int timeoutRelease = FALSE;

void ReleaseClient(int idx_so);

int main(void)
{
   SOCKET Sock_8187, New_Conn;
   struct sockaddr_in Host_addr;
   struct sockaddr_in Client_addr;
   int sin_size;
   int hasConnect, hasMessage;
   int maxSocket, sidx, New_Sidx, numbytes, sidx2;
   char buf[MAXDATASIZE];
   unsigned long pulArgp;
   char *str;
   int tmpcount=1;

   if ((Sock_8187 = socket(AF_Inet,Sock_Stream,0)) == Invalid_Socket)
   {
      perror("socket");
      exit(1);
   }

   Host_addr.sin_family = AF_INET;
   Host_addr.sin_port = htons(Host_Port);
   Host_addr.sin_addr.s_addr = INADDR_ANY;
   memset(&(Host_addr.sin_zero), 0, 8);

   if (bind(sock_8187,(struct sockaddr *)&host_addr,sizeof(struct sockaddr))
   ==socket_error)
   {
      perror("bind");
      exit(1);
   }

   pulArgp = 1;
```

```
if(ioctlsocket(Sock_8187, FIONBIO, &pulArgp))
{
   perror("ioctlsocket");
   exit(1);
}

if (listen(Sock_8187, 5) == SOCKET_ERROR)
{
   perror("listen");
   exit(1);
}

hasMessage = FALSE;
memset(WaitSocketCount, 0, sizeof(WaitSocketCount));
printf("Server started, wait for connect...\n");
while(1)
{
   if (socketTotal > 0)
      hasConnect = Host_WaitForClient(Sock_8187, 0);
   else
      hasConnect = Host_WaitForClient(Sock_8187, 5);

if(hasConnect)
   {
      printf("Receive client connect request...\n");
      sin_size = sizeof(struct sockaddr_in);
      if ((New_Conn = accept(Sock_8187, (struct sockaddr *)&Client_addr, &
            sin_size)) == INVALID_SOCKET)
      {
            perror("accept");
            continue;
      }

      if (New_Conn != INVALID_SOCKET)
         {
            if (socketTotal < Max_Conn)
            {
               remoteSocket[socketTotal] = New_Conn;
               New_Sidx = socketTotal;
               socketTotal++;
            }
            else
            {
```

```
        if(send(new_conn,"connetion full,you are going to be disconnected!\n",
           50, 0) == Socket_Error)
              perror("send");
        closesocket(New_Conn);
        printf("Connetion full, disconnect client!\n");
     }
  }
  else
     printf("(TCP) Invalid incoming socket!\n");

  str = "Hello, world!\n";
  if (send(remoteSocket[New_Sidx],str,strlen(str), 0) == Socket_Error)
     perror("send");

  }

  if(socketTotal>0)
  {
     for(sidx=0; sidx<socketTotal; sidx++)
     {
        hasMessage = Host_WaitForClient(remoteSocket[sidx], 0);
        if(hasMessage)
        {
           if((numbytes=recv(remotesocket[sidx],buf,sizeof(buf),0))==socket_error)
           {
              ReleaseClient(sidx);
           }
           else
           {

              if(numbytes>0)
                 printf("Server receive: %s", buf);

              if(tmpcount%2==0)
                 str = "ACK\n";
              else
                 str = "A C K\n";

              if(numbytes==0)
              {
                 ReleaseClient(sidx);
              }
              else if(send(remotesocket[sidx],str,strlen(str),0)==socket_error)
```

```
                    {
                    ReleaseClient(sidx);
                    }

                    memset(buf, 0, sizeof(buf));
                    tmpcount++;
                    if(tmpcount>100)
                    tmpcount = 1;

                    WaitSocketCount[sidx] = 0;
                  }
                }
                else
                  WaitSocketCount[sidx]++;

                if(WaitSocketCount[sidx]>10000)
                {
                  timeoutRelease = TRUE;
                  ReleaseClient(sidx);
              }
            }
          }
        }

        return 0;
    }

int Host_WaitForClient(int WaitSocket, int i_iWaitMilliSec)
{
    fd_set FdSet;
    struct timeval    waitTime;

    FD_ZERO(&FdSet);
    FD_SET(WaitSocket, &FdSet);
    waitTime.tv_sec    = i_iWaitMilliSec / 1000;
    waitTime.tv_usec = (i_iWaitMilliSec % 1000)*1000L;

    if (select(0, &FdSet, NULL, NULL, &waitTime) > 0)
      return TRUE;
    return FALSE;
}

void ReleaseClient(int idx_so)
{
    int sidx, sidx2;
```

```
    sidx = idx_so;

    if(timeoutRelease)
    {
       if(send(remotesocket[sidx],"Connetion timeout,you are going to be disconnected
          !\n", 53, 0) == -1)
             perror("send");
    }

    if(remoteSocket[sidx]!=INVALID_SOCKET)
    {
       if(closesocket(remoteSocket[sidx])!=0)
             printf("Release client resource fail!");
    }

  for(sidx2 = sidx; sidx2<= socketTotal-1; sidx2++)
    {
       if(sidx2<socketTotal-1)
       {
          WaitSocketCount[sidx2] = WaitSocketCount[sidx2+1];
          remoteSocket[sidx2] = remoteSocket[sidx2+1];
       }
       else if(sidx2==socketTotal-1)
       {
          WaitSocketCount[sidx2] = 0;
          remoteSocket[sidx2] = NULL;
       }

    }

    socketTotal--;

    if(timeoutRelease)
       printf("Connetion timeout, disconnect client %d!\n", sidx);
    else
       printf("Socket error, disconnect client %d!\n", sidx);

    if(socketTotal==0)
       printf("Wait for client connect...\n");

    timeoutRelease = FALSE;

  }
```

TCP_CLIENT.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Server_Port 9510
#define MAXDATASIZE 100
int main(int argc, char *argv[])
{
   SOCKET SO_8187;
   int numbytes=0;
   char buf[MAXDATASIZE];
   struct hostent *he;
   struct sockaddr_in Server_addr;
   char *str1, *str2, *str;
   int tmpcount=1;
   str1 = "TCP\n";
   str2 = "Client\n";
   if (argc != 2)
   {
      fprintf(stderr,"usage: server hostname\n");
      exit(1);
   }
   if ((he=gethostbyname(argv[1])) == NULL)
   {
      perror("gethostbyname");
      exit(1);
   }
   if((SO_8187=socket(af_inet,sock_stream,ipproto_TCP))==incalid_Socket)
   {
      perror("socket");
      exit(1);
   }
   Server_addr.sin_family = AF_INET;
```

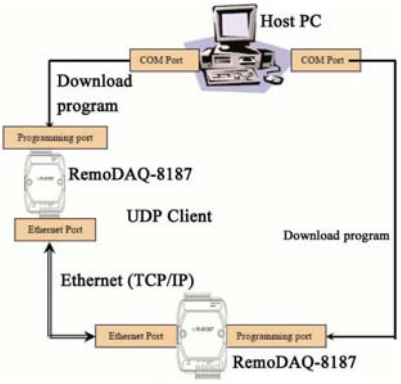
```
    Server_addr.sin_port = htons(Server_Port);
    Server_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(Server_addr.sin_zero), 0, 8);
    if (connect(SO_8187, (struct sockaddr *)&Server_addr, sizeof(struct sockaddr))
       == SOCKET_ERROR)
    {
       perror("connect");
       exit(1);
    }
    while(1)
    {
       if ((numbytes=recv(SO_8187,buf,maxdatasize-1, 0)) == Socket_error)
       {
          perror("recv");
          exit(1);
       }
       if(numbytes>0)
       {
          printf("Received: %s",buf);
          memset(buf, 0, sizeof(buf));
          if(tmpcount%2==0)
             str = str1;
          else
             str = str2;
          sleep(1);
          if (send(SO_8187, str, strlen(str), 0) == SOCKET_ERROR)
          {
             perror("send");
             exit(1);
          }
          tmpcount++;
          if(tmpcount>100)
             tmpcount=1;
       }
       else
       {
          closesocket(SO_8187);
          break;
       }
    }
    return 0;
}
```

4.9 UDP 连接

示例程序：Userver.exe 和 Uclient.exe

源文件：UDP_Server.c 和 UDP_Client.c

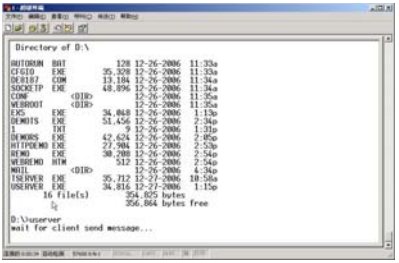
硬件配置：见下图



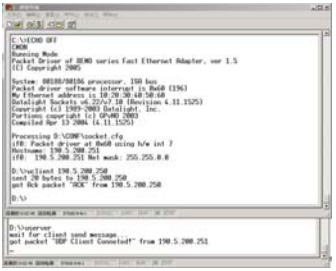
1. 参考有关设置 IP 地址章节设置 RemoDAQ-8187 服务器的 IP 地址

2. 设置 RemoDAQ-8187 客户端的 IP 地址

3. 将 Userver.prj 编译为 Userver.exe 文件，下载到服务端 RemoDAQ-8187 D 盘下
4. 将 Uclient.prj 编译为 Uclient.exe 文件，下载到客户端 RemoDAQ-8187 D 盘下
5. 在服务器端运行 Userver.exe 程序



6. 在客户端运行 Uclient.exe 程序



UDP_Server.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Errno errno
#define FALSE 0
#define TRUE 1
#define Host_Port 9510
#define MAXBUFLEN 100
int main(void)
{
   SOCKET Host_Sock;
   struct sockaddr_in Host_addr;
   struct sockaddr_in Client_addr;
   int hasMessage = FALSE;
   unsigned long pulArgp;
```

```
char buf[MAXBUFLEN];
int addr_len, numbytes;
char* ackmsg = "ACK";
if ((Host_sock=socket(pf_inet,sock_dgram,ipproto_udp)) == invalid_socket)
{
   perror("socket");
   exit(1);
}
Host_addr.sin_family = AF_INET;
Host_addr.sin_port = htons(Host_Port);
Host_addr.sin_addr.s_addr = INADDR_ANY;
memset(&(Host_addr.sin_zero), 0, 8);
if (bind(Host_Sock, (struct sockaddr *)&Host_addr,sizeof(struct sockaddr)) ==
   Socket_error)
{
   perror("bind");
   exit(1);
}
pulArgp = 1;
if(ioctlsocket(Host_Sock, FIONBIO, &pulArgp))
{
   perror("ioctlsocket");
   exit(1);
}
printf("wait for client send message...\n");
while(1)
{
   hasMessage = Host_WaitForMessage(Host_Sock, 0);
   if(hasMessage)
   {
      addr_len = sizeof(struct sockaddr);
      if ((numbytes = recvfrom( Host_Sock, buf, sizeof(buf), 0, (struct sockaddr *)
            &Client_addr, &addr_len)) == SOCKET_ERROR)
      {
         perror("recvfrom");
         if (errno == EWOULDBLOCK)
            printf("EWOULDBLOCK");
         break;
      }
      buf[numbytes] = 0;
      printf("got packet \"%s\" from %s\n", buf, inet_ntoa(Client_addr.sin_addr));
         if ((numbytes=sendto(Host_Sock, ackmsg, strlen(ackmsg), 0,
            (struct sockaddr *)&Client_addr,sizeof(struct sockaddr)))==socket_error)
         {
            perror("sendto");
```

```
                break;
            }
        }
    }
    closesocket(Host_Sock);
    return 0;
}
int Host_WaitForMessage(int serverSocket, int i_iWaitMilliSec)
{
    fd_set FdSet;
    struct timeval    waitTime;
    FD_ZERO(&FdSet);
    FD_SET(serverSocket, &FdSet);
    waitTime.tv_sec    = i_iWaitMilliSec / 1000;
    waitTime.tv_usec = (i_iWaitMilliSec % 1000)*1000L;
    if (select(0, &FdSet, NULL, NULL, &waitTime) > 0)
        return TRUE;
    return FALSE;
}
```

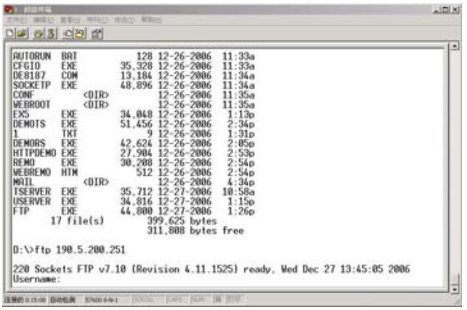
UDP_Client.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Errno errno
#define BufferSize    100
#define Host_Port 9510
int main(int argc, char *argv[])
{
    SOCKET SO_8187;
    struct sockaddr_in Server_addr;
    struct sockaddr_in From_Addr;
    struct hostent *he;
    char buf[BufferSize];
    int numbytes;
```

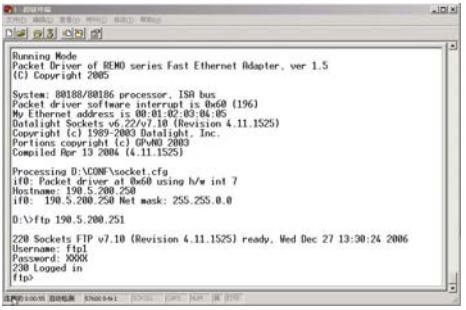
```
unsigned int From_Size;
char* msg = "UDP Client Conneted!";
if (argc != 2)
{
   fprintf(stderr,"usage: uclient xxx.xxx.xxx.xxx\n");
   exit(1);
}
if ((he=gethostbyname(argv[1])) == NULL)
{
   perror("gethostbyname");
   exit(1);
}
if ((SO_8187=socket(af_inet,Sock_dgram,ipproto_udp))==invalid_socket)
{
   perror("socket");
   exit(1);
}
Server_addr.sin_family = AF_INET;
Server_addr.sin_port = htons(Host_Port);
Server_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(Server_addr.sin_zero), 0, 8);
if ((numbytes=sendto(SO_8187,msg,strlen(msg),0,(struct sockaddr *)& Server_
   addr, sizeof(struct sockaddr))) == SOCKET_ERROR)
{
   perror("sendto");
   exit(1);
}
printf("sent %d bytes to %s\n", numbytes, inet_ntoa(Server_addr.sin_addr));
From_Size=sizeof(from_addr);
if((numbytes=recvfrom(so_8187,buf,sizeof(buf),0,(struct sockaddr *)&From_Addr,
   &From_Size)) == -1)
{
   perror("recvfrom");
   exit(1);
}
buf[numbytes] = 0;
printf("got Ack packet \"%s\" from %s\n", buf, inet_ntoa(From_Addr.sin_addr));
closesocket(SO_8187);
return 0;
}
```

4.10 FTP 客户端

1. 下载 FTP.exe (在 RemoDAQ-8187\Source\Drive D\Extnsion files\webroot)文件到 D 盘的“webroot”目录下
2. 进入“Webroot”目录下，运行 FTP.EXE + FTP 服务器地址



3. 通过输入用户名和密码登陆 FTP 服务器，然后您就可以通过输入 FTP 命令来访问 FTP 服务器。



第 5 章 编程和函数库

5.1 引言

用户可以使用RemoDAQ-8187提供的函数库设计用户的应用程序。RemoDAQ-8187提供了 10 个小的函数库来有效的利用 RemoDAQ-8187 的内存空间。

5.1.1 关于 RemoDAQ-8187 控制器的编程细节

RemoDAQ-8187 控制器的操作系统是 ROM-DOS。它允许用户运行和使用汇编语言或其他语言如 C 或 C++编写的应用程序。为了成功的编写应用程序，请留意下面的意见和一些限制条件。

5.1.2 Mini BIOS 功能

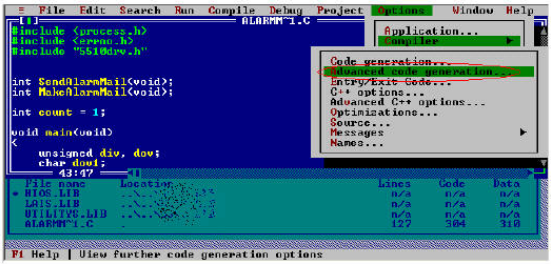
RemoDAQ-8187 控制器的 mini BIOS 提供了 10 个功能函数供用户调用。另外，特定的编程语言例如 QBASIC 对 BIOS 功能的直接调用可能无法执行。

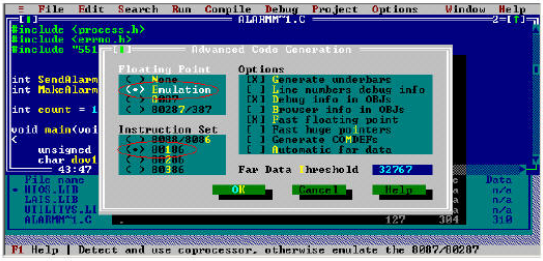
RemoDAQ-8187 控制器的 mini BIOS 功能调用见下表所示：

函数	子函数	功能	函数	子函数	功能
07h		186 或更高级的CD机取消指令	18h		输出 failed to boot ROMDos 信息
10h	0eh	TTY输出清零	19h		重新启动系统
11h		获取设置	1ah	0	获取秒
12h		获取内存大小		1	设置秒
15h	87h	读取扩展内存		2	获取时间
	88h	扩展内存大小		3	设置时间
	C0h	PS/2或AT格式A20门控列表		4	获取日期
16h	0	读TTY字符		5	设置日期
	1	获取TTY状态	1ch		定时
	2	获取TTY标志			

5.1.3 编译程序代码

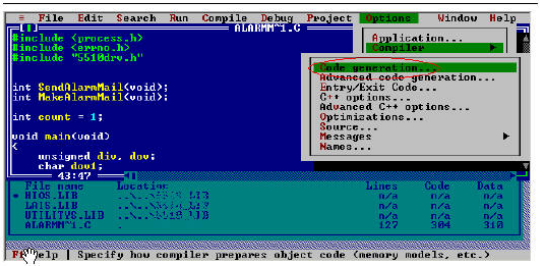
RemoDAQ-8187 使用一个 80188 的 CPU，所以在程序被下载到闪存之前必须被编译成 80188 或 80186 兼容的代码。如果您使用的是 Borland C 编译器，您可以按照以下方式来编译程序：

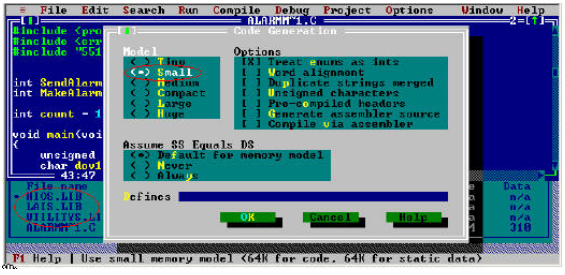




5.1.4 不同的内存模式对应不同的库类型

RemoDAQ-8187 函数库支持四种内存模式：Small，Medium，Compact 和 Large。您可以使用适合您内存模式的函数库。





5.1.5 限制

RemoDAQ-8187 控制器提供了 1MB 的闪存作为 D 盘。其中有 960KB 的空间可以存储用户的应用程序。有一些附加的文件和网络程序占用了 D 盘，所以留给用户使用的空间不到 960KB。另外，还有 256KB 的闪存和 384KB 的 SRAM 可以被用户通过函数库来访问。

5.1.6 看门狗定时器的编程

如果您想使用看门狗功能，必须通过调用函数使它使能。当它被使能之后，每隔最多 1.6 秒被清一次，否则它将复位 CPU，或者触发一个非屏蔽中断。在程序最后，您需要调用函数将看门狗功能禁止。

5.2 函数库的类别

RemoDAQ-8187 控制器有以下 10 类函数库：

- 类别 A.系统功能（UTILITY*.LIB）
- 类别 B.I/O 模块功能（HIO*.LIB）
- 类别 C.通讯功能（A4_COMM*.LIB）
- 类别 D.MODBUS/RTU 功能（RTU*.LIB）
- 类别 E.MODBUS/TCP 功能（MBTCP*.LIB）
- 类别 F.网络套接字功能（SOCKET*.LIB）
- 类别 G.HTTP 功能（CGI_LIB*.LIB）

注：1 这些函数库只支持 DOS 下的 Borland C 3.0

2 请包含所有必须的 RemoDAQ-8187 函数库

5.2.1 类别 A：系统功能（UTILITY*.LIB）

dsl_printf()
REMOdelay
Get_BoardID
Get_NodeID
GetRTCtime
SetRTCtime
EraseSector
ProgramByte
ProgramSector
read_mem
Get_SysMem
Set_SysMem
Get_NVRAM_Size
Set_NVRAM_Size
write_backup_ram
read_backup_ram
Timer_Init
Timer_Reset
Timer_Set
Release_All
tmArriveCnt
WDT_clear
WDT_disable
WDT_enable
display_inti()

display_digit()
BatteryStatus()
Ver()

5.2.2 类别 B：I/O 模块功能（HIO*.LIB）

InitDIFilter()
GetDIO()
SetDIO()
Ver_HIOLib()

5.2.3 类别 C：通讯功能：（A4_COMM*.LIB）

SIO_Open()
SIO_Close()
SIO_SetState()
SIO_RecvBytes()
SIO_SendBytes()
SIO_GetAvaiRecvBytes()
SIO_GetAvaiSendBuf()
SIO_PurgeBuf()
SIO_MakeCheckSum()
SIO_MakeCRC16()
SIO_Carrier()
SIO_ClearBreak()
SIO_SetBreak()
SIO_GetLineStatus()
SIO_SetLineParams()
SIO_GetModemStatus()
SIO_LowerRaise_RTS_DTR()
SIO_ModemInitial()
SIO_ModemAutoanswer()
SIO_ModemCommand()
SIO_ModemCommand_State()
SIO_ModemDial()
SIO_ModemHandup()
Ver_COMLib()

5.2.4 类别 D：MODBUS/RTU 功能（RTU*.LIB）

Modbus_COM_Init()
Modbus_COM_Release()
Error_Code()

REMORTU_ForceMultiCoils()
REMORTU_ForceSingleCoil()
REMORTU_PresetMultiRegs()
REMORTU_PresetSingleReg()
REMORTU_ReadCoilStatus()
REMORTU_ReadHoldingRegs()
REMORTU_ReadInputRegs()
REMORTU_ReadInputStatus()
REMORTU_ModServer_Create()
Ver_RTU_Mod()

5.2.5 类别 E：MODBUS/TCP 功能（MBTCP*.LIB）

Ver_TCP_Mod()

Modbus TCP 客户机功能：

ReturnErr_code()
REMOTCP_Connect()
REMOTCP_Disconnect()
REMOTCP_ForceMultiCoils()
REMOTCP_ForceSingleCoil()
REMOTCP_PresetMultiRegs()
REMOTCP_PresetSingleReg()
REMOTCP_ReadCoilStatus()
REMOTCP_ReadHoldingRegs()
REMOTCP_ReadInputRegs()
REMOTCP_ReadInputStatus()

Modbus TCP 服务器功能：

REMOTCP_ModServer_Create()
REMOTCP_ModServer_Update()
REMOTCP_ModServer_Release()

5.2.6 类别 F：网络套接字功能（SOCKET*.LIB）

套接字功能：

accept ()
bind ()
closesocket ()
connect ()
ioctlsocket ()
getpeername ()
getsockname ()

getsockopt ()
htonl ()
htons ()
inet_addr ()
inet_ntoa ()
listen ()
ntohl ()
ntohs ()
recv ()
recvfrom ()
select ()
send ()
sendto ()
setsockopt ()
shutdown ()
socket ()

数据库功能:
gethostbyaddr()
gethostbyname()
gethostname ()
getservbyport()
getservbyname()
getprotobynumber()
getprotobyname()

5.2.7 类别 G：HTTP 功能(CGI_LIB*.LIB)

套接字功能:
HttpRegister()
HttpDeRegister()
HttpGetData()
HttpSendData()
HttpSubmitFile()
HttpGetStatus()
HttpGetVersion()
GetStackPointer()
GetStackSegment()
SetStackPointer()
SetStackSegment()

5.3 函数库描述

5.3.1 系统函数（UTILITY*.LIB）

dsl_printf

语法：

void dsl_printf(char *pFormat,…);

描述：

在上位机上显示字符串。这个函数和 printf()的用法相同。然而它的优先权比较低。

参数：

具有和 printf()相同的标准 Borland C 3.0 函数库。

返回值：

无

示例：

```
#include”8187drv.h”
void main(void)
{
dsl_printf(“Hello,this is for test.”);
}
```

说明：

如果 printf()函数用在 Modbus/RTU 服务器“while”循环里面，它将降低服务器的性能。所以我们强烈建议使用优先权更低的 dsl_printf()函数。

Remodelay

语法：

void REMOdelay(unsigned short msec)

描述：

将程序的运行延时一个指定的时间

参数	**描述**
毫秒	0 ~ 65535

返回值：

无

示例：

```
#include”8187drv.h”
void main(void)
{
/*codes placed here by user */
REMOdelay(1000); /*delay 1 sec.*/
/*codes placed here by user */
}
```

备注：Remodelay 可能回降低运行的性能，所以推荐使用循环来代替。

Get_BoardID

语法：

unsigned char Get_BoardID(void)

描述：

获得 I/O 板的 ID 号

参数：

无

返回值：

返回 I/O 板的 ID 号

备注：

无

Get_NodeID

语法：

unsigned char Get_NodeID(void)

描述：

获得 RemoDAQ-8187 控制器的 ID 号

参数：

无

返回值：

返回 RemoDAQ-8187 的 ID 号

示例：

```
#include “8187drv.h”
void main(void)
{
        int i,j;
        unsigned char mID,found;
  dsl_printf(“\n Welcome to REMO8187 PC-Based Controller”);
  dsl_printf(“\n Scan I/O module …”);
  dsl_printf(“\n REMO8187 NodeID=%02Xh”,Get_NodeID());
  /*Scan REMO8187 Slot IO Module */
  mID=Get_BoardID();
  found=0;
        if(mID==EB50_ID)
        {
          dsl_printf(“\n Slot=EB50”);
          found=1;
        }
  if(found ==0) dsl_printf(“\n Slot=None installed”);
}
```

备注：无

GetRTCtime；SetRTCtime

语法：

unsigned char GetRTCtime(unsigned char Time)

void SetRTCtime(unsigned char Time,unsigned char data)

描述：

GetRTCtime：读时钟

SetRTCtime：设置时间和日期

参数	**描述**	
时间	RTC_sec	秒
	RTC_min	分
	RTC_hour	小时
	RTC_day	日期
	RTC_week	星期
	RTC_year	年
日期	新日期	

返回值：

和用户的命令有关

示例：

```
#include”8187drv.h”
void main(void)
{unsigned char sec=0,min=0,hour=12;
   dsl_printf(“Time %02d:%02d:%02d\n”,GetRTCtime(RTC_hour),
   GetRTCtime(RTC_min),GetRTCtime(RTC_sec));
   dsl_printf(“Set current time 12:00:00\n”);
   SetRTCtime(RTC_sec,sec);
   SetRTCtime(RTC_min,min);
   SetRTCtime(RTC_hour,hour);
   Dsl_printf(“Time %02d:%02d:%02d\n”,GetRTCtime(RTC_hour);
   GetRTCtime(RTC_min),GetRTCtime(RTC_sec));
}
```

备注：

无

EraseSector；ProgramByte；ProgramSector

语法：

unsigned short EraseSector(unsigned long ulBase)

unsigned short ProgramByte(unsigned long ulAddress, BYTE byte)

unsigned short ProgramSector(unsigned long ulAddress_s, unsigned char far * SECTOR_DATA)

描述：

EraseSector：擦除 256KB 闪存中的 64KB 字节数据。

ProgramByte：向 256KB 闪存中写入一个字节的数据。这个命令支持数据存储或大量信息的存储。

ProgramSector：通过一个全局变量 SECTOR_DATA[]来将一个完整的 32KB 扇区，写入 256KB 闪存中。

参数	描述
ulBase	删除用户确定的一个地址段
ulAddress	以字节方式传输数据到指定的目的地址
byte	用户想写入闪存中的具体数据
ulAddress_s	用户确定的闪存地址
SECTOR_DATA	用户数组的起始地址指针

返回值：

1 成功传输数据到闪存

0 传输过程中出现错误

Read_mem

语法：

unsigned char read_mem(int memory_segment,unsigned int i)

描述：

读 256KB 闪存的远程存储器。

参数	描述
memory_segment	用户在 0X8000 到 0XBF00 区间定义的地址
i	用户在地址区间 0x0000 到 0x0FFF 的偏移量

返回值：

指定地址的数据

示例：

```
#include "8187drv.h"
void main(void)
{
  unsigned char sector[32768];
  unsigned char data;
  unsigned long addr,sector_num;
  unsigned int i;
  dsl_printf("erase sector 0x80000L\n");
  if(EraseSector(0x80000L))
    dsl_printf("erase succeed \n");
  dsl_printf("Write data(55) to 0x80000~0x80001\n");
  data=55;
  ProgramByte(0x80000L,data);
  ProgramByte(0x80000L+1,data);
```

```
        ProgramByte(0x80000L+2,data);
        for(i=0;i<3;i++)
        {
        dsl_printf("read%d data=%d\n",i,read_mem(0x8000,0x0000+i));
        }
        dsl_printf("erase sector 0x80000L\n");
        if(EraseSector(0x80000L))
           dsl_printf("erase succeed \n");
        data = 1;
        for(i=0;i<32768;i++)
           *(sector+i)=data;
        dsl_printf("Write data(0x01) to 0x80000~0x87FFF\n");
        ProgramSector(0x80000,&sector);
        for(i=0;i<100;i++)
        {
        dsl_printf("read%d data=%d\n",i,read_mem(0x8000,0x0000+i));
        }
    }
```

备注：

无

Get_SysMem；Set_SysMem

语法：

unsigned char Get_SysMem(unsigned char which_byte)

void Set_SysMem(unsigned char which_byte, unsigned char data)

描述：

Get_SysMem：从 SRAM 中读一个字节

Set_SysMem：向 SRAM 中写一个字节

参数	**描述**
which_byte	从 0-112，用户定义的
data	要保存的数据

返回值：

GetSysMem：返回 SRAM 中存放的数据

Set_SysMem：无

示例：

```
    #include "8187drv.h"
    void main(void)
    {
       unsigned char data[4] = {1,2,3,4};
       int i;
       /* save current value */
```

```
    for(i=10;i < 14;i++)
    {
      Set_SysMem(i, data[i-10]);
      dsl_printf("data=%d\n",Get_SysMem(i));
    }
  }
```

备注：

无

Get_NVRAM_Size；Set_NVRAM_Size

语法：

unsigned char Get_NVRAM_Size(void)

void Set_NVRAM_Size(unsigned char sector)

描述：

Get_NVRAM_Size：获得电池后备 RAM 的大小

Set_NVRAM_Size：设置电池后备 RAM 的大小

参数	**描述**
Sector	NVRAM 扇区大小为 4KB，从 1 到 96 扇区

返回值：

Get_NVRAM_Size：扇区已设置的 NVRAM 扇区号

Set_NVRAM_Size：无

示例：

```
  #include "8187drv.h"
  void main()
  {
    unsigned char sector;
    sector = Get_NVRAM_Size();
    dsl_printf(“Backup ram=%dKbyte\n”,sector*4);
    /*Set Bacup ram 40Kbyte*/
    Set_NVRAM_Size(10);
  }
```

备注：

无

Write_backup_ram；Read_backup_ram

语法：

void write_backup_ram(unsigned long index,unsigned char data)

unsigned char read_backup_ram(unsigned long index)

描述：

write_backup_ram：写一个字节到电池后备存储器中

read_backup_ram：读电池后备 RAM 中的值

参数　　描述

index　数据在电池后备 Ram 中的一个索引,0~393214;384KB(Max)电池后备 SRam

data　　要写入电池后备 SRAM 中的数据

返回值：

write_backup_ram：无

read_backup_ram：在索引地址中的一个字节数据

示例：

```
#include "8187drv.h"
void main()
   {
   unsigned long addr;
   unsigned char data;
   /*write the data 0x55 into battery backup memory, index=10*/
   data=0x55;
   write_backup_ram(10,data);
   dsl_printf("data=%x\n",read_backup_ram(10));
}
```

备注：

无

Timer_Init

语法：

int Timer_Init()

描述：

初始化 80188 的定时器，返回 0 表示初始化成功，返回 1 表示定时器已被初始化。

参数：

无

返回值：

0：初始化成功

1：定时器已经被初始化了

备注：

无

示例：

参考 Release_All

Timer_Reset

语法：

void Timer_Reset(int idx)

描述：

复位定时器标志位为初始化状态

参数	**描述**
idx	定时器 ID

返回值：

无

备注：

无

示例：

参考 Release_All

Timer_Set

语法：

int Timer_Set(unsigned int msec)

描述：

申请处理器的定时器，并设置定时器的时间间隔。定时器的时间间隔是以 5 毫秒为增量的。如果函数成功执行则返回一个代表这个定时器的一个十进制 ID 号。

返回值如果为“-1”则表示函数执行有错误。用户需要确定是否指定的这个定时器时间溢出。

变量“tmArriveCnt[idx]的值可以用来确定定时器的状态。0 表示定时器正在计数，非 0 的值表示定时器超时。

参数	**描述**
msec	时间间隔设置，最大值为 65536。

返回值：

0~100	成功。返回值为定时器 ID
-1	函数错误

备注：

RemoDAQ-8187 的定时器函数是仿照 PLC 中的定时器功能来设置的。如果应用程序在同一时间使用多个计数器那么将使程序的运行效率降低。

示例：

参考 Release_All

Release_All

语法：

void Release_All()

描述：

释放 RemoDAQ-8187 定时器的所有资源

参数：

无

返回值：

无

备注：

无

示例：

```
#include "8187drv.h"
void main()
{
   int idx;
   /* Initializes the timer built into the 80188 microprocessor */
   Timer_Init();
   /* Sets time interval of the timer to 1 second. */
   idx=Timer_Set(1000);
   /* Checks whether the timer has timed out */
   while(tmArriveCnt[idx]==0)
   {
      /* user can attend to other tasks... */
      dsl_printf("test");
   }
   /* Resets the current timer to its initial state. */
   Timer_Reset(idx);
   /* Releases all timer resources */
   Release_All();
}
```

WDT_clear；WDT_disable；WDT_enable

语法：

void WDT_clear(void)

void WDT_disable(void)

void WDT_enable(void)

描述：

WDT_clear：清看门狗定时器

WDT_disable：禁止看门狗定时器

WDT_enable：允许看门狗定时器

如果看门狗被允许，那么您需要最多间隔 1.6 秒清一下看门狗，否则系统将被重启。看门狗的默认设置是禁止的。

参数：无

返回值：无

示例：

```
#include "8187drv.h"
```

```
void main(void)
{
  int i;
  WDT_enable();
  for(i=0;i<100;i++)
  {
    /*put your code in Here*/
    WDT_clear();
    /*put your code in Here*/
  }
  WDT_disable();
}
```

备注：

无

BatteryStatus

语法：

int BatteryStatus(void)

描述：

检测用于电池备份的电池状态

参数：

无

返回值：

0 电池电量不足，需要更换新电池

1 电池状态正常

Ver

语法：

void Ver(char *vstr)

描述：

检测函数库的版本号。

参数	**描述**
vstr	版本信息号

返回值：

无

示例：

```
void main(void)
{
  Ver(library_ver);
  dsl_printf("The version of utility library is %s\n", library_ver);
}
```

5.3.2 I/O 模块功能（HIO*.LIB）

InitDIFilter

语法：

void initDIFilter(int iCh,unsigned int MIN_Lo_Width,unsigned int MIN_High_Width)

描述：

设置 DI 通道的数字滤波时间间隔

参数	**描述**
iCh	通道号（0~4）
MIN_Lo_Width	低电平状态的 DI 滤波时间间隔
MIN_High_Width	高电平状态的 DI 滤波时间间隔

返回值：

无

备注：

参考值

时间间隔	频率
15ms	50Hz
30ms	20Hz
50ms	12Hz

GetDIO；SetDO

语法：

Uchar GetDio(uchar I_ucModuleID,uchar I_ucMode,uchar I_ucChannel,Ulong*o_ulValue)

Uchar SetDO(Uchar I_ucModuleID,Uchar I_ucMode,Uchar I_ucChannel,Ulong I_ulValue)

描述：

读/写 DIO 通道的值

参数	**描述**
i_ucModuleID	模块 ID.
i_ucMode	单个通道或所有通道.
i_ucChannel	单个通道：使用的通道号； 所有的通道 无意义
o_ulValue	从 DIO 读回的值.
i_ulValue	DIO 模块写入的值

返回值：

如果成功，返回 0，如果出现错误，返回值如下：

Illegal_Setting	-5
Board_Not_Exist	-7

备注：

无

Ver_HIOLib

语法：

void Ver_HIOLib(char *vstr)

描述：

获得 HIO 函数库的版本号.

参数	**描述**
vstr	代表 HIO 版本号的指针

返回值：

无

示例：

```
char library_ver[20];
void main(void)
{
   Ver_HIOLib(library_ver);
   dsl_printf("The version of library is %s\n", library_ver);
}
```

备注：

无

DIO 示例：

```
#include "8187drv.h"
void main()
{
   int tmpCnt=0;
   char c;
   unsigned char type;
   unsigned long div, dov;
   char Ver_Str[30];
   Ver_HIOLib(Ver_Str);
   printf("The HIO library version is %s\n", Ver_Str);
   /* ---- First scan for the existing IO modules -------*/
   type = Get_BoardID();
   /*----Show the module type of each slot on the screen ---*/
   if( type == EB50_ID)
      printf("IO slot is EB50\n");
   else
      printf("IO slot is Null\n");
   printf("press any key to continue...\n");getch();
   /*--- Digital I/O modules don't need to be initialized ---*/
   /*---- Forever loop until the user press any key */
   while(1)
```

```
    {
        tmpCnt++;
        /*--- Set DO Value ------*/
        printf("\n\nSet and Get All Channels Test:\n");
        if((tmpCnt%2)==0)
            dov=0;
        else
        dov=0xf;
        if(SetDO(EB50_ID, AllChannels, 0, dov)==0)
        {
            printf("Set DO value 0x%X, ", dov);
            printf("press any key to continue..\n");getch();
        if(GetDIO(EB50_ID, AllChannels, 0, &div)==0)
            {
                printf("DI value is 0x%X\n", div);
            }
            else
                printf("Get DI failed\n");
        }
        else
            printf("Set DO failed!\n");
        printf("\n\nSet and Get Sigle Channel Test:\n");
        if((tmpCnt%2)==0)
            dov=0;
        else
            dov=1;
        if(SetDO(EB50_ID, SingleChannel, 1, dov)==0)
        {
            printf("Set DO channel 1 value %X, ", dov);
            printf("press any key to continue..\n");getch();
            if(GetDIO(EB50_ID, SingleChannel, 1,&div)==0)
            {
                printf("Channel 1 DI value is %x\n",div);
            }
            else
                printf("Get DI failed\n");
        }
        else
            printf("Set DO failed!\n");
        printf("press 'Q' to quit, the other key to continue..\n");
        c=getch();
        if( c == 'q' || c == 'Q')
            break;
    }
}
```

5.3.3 通信函数库（A4_COMM*.LIB）

SIO_OPEN

语法：

CHAR SIO_Open(UCHAR i_ucPort)

描述：

在其它函数调用 COM 口之前初始化 COM 口。

参数　　描述

I_ucPort　　要初始化的 COM 口

返回值：

如果返回 0 表示成功。如果错误，返回值如下：

COM_already_installed　　-1

Err_Access_COM　　-2

No_Such_Port　　-3

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_Close

语法：

CHAR SIO_Close(UCHAR I_ucPort)

描述：

释放 COM 口占用的资源

参数　　描述

I_ucPort　　要释放的 COM 口

返回值：

如果返回 0 表示成功，如果错误返回值如下：

No_Such_Port　　-3

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_SetState

语法：

CHAR SIO_SetState(UCHAR i_ucPort, ULONG i_ulBaudRate,
UCHAR i_ucParity, UCHAR i_ucDataBits, UCHAR i_ucStopBits)

描述：

设置 COM 口的参数

参数	**描述**
i_ucPort	COM 口号.
i_ulBaudRate	波特率
i_ucParity	奇偶校验
i_ucDataBits	数据位
i_ucStopBits	停止位

返回值：

返回 0 表示成功，错误返回值如下：

No_Such_Port	-3
COM_Not_Installed	-4
Illegal_Setting	-5

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4
i_ucParity	0x00	无校验
	0x08	奇校验
	0x18	偶检验
i_ucDataBits	0x00	5 个数据位
	0x01	6 个数据位
	0x02	7 个数据位
	0x03	8 个数据位
i_ucStopBits	0x00	1 个停止位
	0x04	2 个停止位

SIO_RecvBytes

语法：

INT SIO_RecvBytes(UCHAR i_ucPort, UCHAR i_ucMode, UINT i_uinBytes, UCHAR * o_ucDataBuf)

描述：

从指定 COM 口获取数据

参数	**描述**
I_ucPort	COM 口号
I_ucMode	是否使用 BLOCK 模式
I_uinBytes	读取的字节数
O_ucDataBuf	读取的数据

返回值：

如果成功，则返回读取的字节数，如果出现错误，返回值如下：

No_Such_Port -3

Illegal_Setting -5

RequestOverQueueSize -6

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4
i_ucMode	0x01	Block 模式
	0x02	非 Block 模式

SIO_SendBytes

语法：

INT SIO_SendBytes(UCHAR I_ucPort,UINT I_uinBytes,UCHAR *I_ucDataBuf)

描述：发送数据到指定的 COM 口

参数 **描述**

I_ucPort 端口号

I_uinBytes 要发送的字节数

I_ucDataBuf 发送数据缓存

返回值：

如果成功，返回发送的字节数，如果错误返回值如下：

No_Such_Port -3

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

GetAvaiRecvBytes；SIO_GetAvaiSendBuf

语法：

INT SIO_GetAvaiRecvBytes(UCHAR i_ucPort)

INT SIO_GetAvaiSendBuf(UCHAR i_ucPort)

描述：

SIO_GetAvaiRecvBytes： 返回输入缓存中的字节数

SIO_GetAvaiSendBuf： 返回发送缓存中的字节数

参数 **描述**

I_ucPort 要查看的端口号

返回值：

SIO_GetAvaiRecvBytes：

如果成功，返回在输入缓存中的字节数。如果失败，返回值如下：

No_Such_Port -3

SIO_GetAvaiSendBuf：

如果成功，返回在发送缓存中的字节数。如果失败，返回值如下：

No_Such_Port -3

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_PurgeBuf

语法：

CHAR SIO_PurgeBuf(UCHAR i_ucPort, UCHAR i_ucFlag)

描述：

用户可以清除变量 I_ucFlag 表示的输入缓存和输出缓存

参数	**描述**
I_ucPort	端口号
I_ucFlag	要清除的缓存（输入还是输出）

返回值：

如果成功，返回 0，如果失败，返回值如下：

No_Such_Port -3

Illegal_Setting -5

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4
i_ucFlag	0x01	清输入缓存
	0x02	清输出缓存

COM 口编程示例：

```
#ifdef Product_5510
  #include "5510drv.h"
#elif defined Product_8187
  #include "8187drv.h"
#endif
#include <conio.h>
```

```
#include <dos.h>
#define DataLen 50
#define RecvDataLen 30
void main()
{
   UCHAR tmpCOM, Idx=0;
   CHAR Data[DataLen];
   UCHAR Mode=0; //0=>Receive, 1=>Send
   CHAR VerStr[30];
   UCHAR RecvMode;
   UCHAR nByte=0, nTotalByte=0;
   printf("Arthur New COM Port Library Test 3.3\n");
   printf("================================\n\n");
   Ver_COMLib(VerStr);
   printf("Com Library Version: %s\n", VerStr);
   printf("Enter receive mode( 1: Block mode, 0: unBlock mode):");
   scanf("%d", &tmpCOM);
   if(tmpCOM==0)
      RecvMode=UnBlock_Mode;
   else
      RecvMode=Block_Mode;
#ifdef Product_5510
   printf("Enter COM port selection( 1: COM1, 2: COM2, 3:COM4)");
   scanf("%d", &tmpCOM);
   if(tmpCOM==1)
      tmpCOM=COM1;
   else if(tmpCOM==2)
      tmpCOM=COM2;
   else if(tmpCOM==3)
      tmpCOM=COM4;
#elif defined Product_8187
   printf("Enter COM port selection( 1: COM1, 2: COM2, 3: COM3,4: COM4)");
   scanf("%d", &tmpCOM);
   if(tmpCOM==1)
      tmpCOM=COM1;
   else if(tmpCOM==2)
      tmpCOM=COM2;
   else if(tmpCOM==3)
      tmpCOM=COM3;
   else if(tmpCOM==4)
      tmpCOM=COM4;
#endif
   else
   {
      printf("Wrong selection!\n");
```

```
        return;
    }
    printf("Opening COM Port with 57600 baud...\n");
    if(SIO_Open(tmpCOM)!=0)
    {
        printf("error\n");
        return;
}
if(SIO_SetState(tmpCom,(unsigned long)57600, NO_Parity,data,top1)!=0)
{
    printf("Set State Error\n");
    return;
}
SIO_PurgeBuf(tmpCOM, Clear_RXBuffer);
SIO_PurgeBuf(tmpCOM, Clear_TXBuffer);
while(1)
{
        if(Mode==0)
        {
            if(RecvMode==Block_Mode)
            {
    if(SIO_GetAvaiRecvBytes(tmpCOM)>=RecvDataLen)
                {
                    if(sio_recvbytes(tmpCom,block_mode,recvdatalen, data)==recvdataLen)
                    {
                        if(Data[RecvDataLen-1]==0x0d)
                        {
                            Mode=1;
                            Idx=0;
                        }
                        else
                        {
                            printf("Error Receive\n");
                            return;
                        }
                    }
                    else
                    {
                        printf("Error Receive\n");
                        return;
                    }
                }
                else
                {
                    //do something else here
```

```
            }
        }
        else if(RecvMode==UnBlock_Mode)
        {
            while(nTotalByte<RecvDataLen)
            {
    if((nbyte=sio_recvbytes(tmpCom,unblock_mode,recvdatalen,&data[ntotalbyte]))>=0)
                {
                    nTotalByte+=nByte;
                }
                else
                {
                    printf("Error Receive\n");
                    return;
                }
            }
            Mode=1;
            Idx=0;
            nTotalByte=0;
        }
    }
    else
    {
      if(SIO_GetAvaiSendBuf(tmpCOM)>=RecvDataLen)
      {
        if(SIO_SendBytes(tmpCOM,RecvDataLen, Data)==RecvDataLen)
        {
            Idx=RecvDataLen;
            if(Data[Idx-1]==0x0d)
                {
                    Mode=0;
                    Idx=0;
                }
        }
        else
        {
            printf("Error Send\n");
            return;
        }
      }
        else
            {
                printf("do something send\n");
                //do something else...
            }
```

```
        }
      }
    SIO_Close(tmpCOM);
    }
```

SIO_MakeCheckSum

语法：

UINT SIO_MakeCheckSum(UCHAR * i_ucDataBuf, UINT i_uiLen)

描述：

计算字符串或数组的校验和

参数	**描述**
I_ucDataBuf	要计算校验和的字符串或数组
I_uiLen	字符串或数组的长度

返回值：

计算出的校验和

备注：

无

SIO_MakeCRC16

语法：

UINT SIO_MakeCRC16(UCHAR * i_ucDataBuf, UINT i_uiLen)

描述：

计算字符串的 16 位 CRC。

参数	**描述**
I_ucDataBuf	要计算 CRC 的字符串
I_uiLen	字符串长度

返回值：

CRC 的 16 位代码

备注：

无

SIO_Carrier

语法：

CHAR SIO_Carrier(UCHAR i_ucPort)

描述：

检测 COM 口的载波信号

参数	**描述**
I_ucPort	端口号

返回值：

如果成功，返回 1，如果失败，返回值如下：

FALSE	0
Illegal_Setting	-5

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_ClearBreak；SIO_SetBreak

语法：

CHAR SIO_ClearBreak(UCHAR i_ucPort)

CHAR SIO_SetBreak(UCHAR i_ucPort)

描述：

SIO_ClearBreak：清指定端口的 BREAK 信号.

SIO_SetBreak：在指定端口发送 BREAK 信号

参数	**描述**
I_ucPort	端口号

返回值：

如果成功，返回 0，如果失败，返回值如下：

Illegal_Setting	-5

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_GetLineStatus；SIO_SetLineParams；SIO_GetModemStatus

语法：

CHAR SIO_GetLineStatus(UCHAR i_ucPort)

CHAR SIO_SetLineParams(UCHAR i_ucPort, UCHAR i_ucParams)

CHAR SIO_GetModemStatus(UCHAR i_ucport)

描述：

SIO_GetLineStatus：读指定端口的线路控制寄存器

SIO_SetLineParams：写指定端口的线路控制寄存器

SIO_GetModemStatus：读指定端口的 modem 状态寄存器

参数	**描述**
i_ucPort	端口号

i_ucParams　　UART 寄存器参数

返回值：

如果成功，返回 UART 寄存器值。如果失败，返回如下：

Illegal_Setting　　-5

备注：

参数	值	描述
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_LowerRaise_RTS_DTR

语法：

Char Sio_lowerraise_RTS_DTR(Uchar i_ucport,Uchari_ucl_r_mode,Uchar i_ucSignal)

描述：

RTS/DTR 信号置高或置低

参数　　描述

I_ucPort　　端口号

I_ucL_R_Mode　　置高或置低

I_ucSignal　　RTS/DTR

返回值：

如果成功返回 0，如果失败返回如下：

Illegal_Setting　　-5

备注：

参数	值	描述
i_ucPort	COM1	COM1
i_ucL_R_Mode	0x01	置高
	0x02	置低
i_ucSignal	0x01	RTS 信号
	0x02	DTR 信号

SIO_ModemInitial

语法：

CHAR SIO_ModemInitial(UCHAR i_ucPort)

描述：

将 modem 设置为初始化状态

参数　　描述

I_ucPort　　端口号

返回值：

如果成功，返回 0，如果失败返回如下：

Illegal_Setting　　-5

备注：

参数	值	描述
i_ucPort	COM1	COM1

SIO_ModemAutoanswer

语法：

CHAR SIO_ModemAutoanswer(UCHAR i_ucPort)

描述：

设置 modem 为自动应答模式。

参数　　描述

I_ucPort　　端口号

返回值：

如果成功返回 0，如果失败返回值如下：

Illegal_Setting　　-5

备注：

参数	值	描述
i_ucPort	COM1	COM1

SIO_ModemCommand

语法：

CHAR SIO_ModemCommand(UCHAR i_ucPort, UCHAR * i_ucCmdStr)

描述：

发送 AT 命令到 modem。

参数　　描述

I_ucPort　　端口号

I_ucCmdStr　　命令字符串

返回值：

如果成功返回 0，错误时返回值如下：

Illegal_Setting　　-5

备注：

参数	值	描述
i_ucPort	COM1	COM1

SIO_ModemCommand_State

语法：

CHAR SIO_ModemCommand_State(UCHAR i_ucPort)

描述：

将 modem 从数据模式设置为命令模式。

参数　　描述

I_ucPort　　端口号

返回值：

如果成功返回 0，错误时返回值如下：

Illegal_Setting -5

备注：

参数	值	描述
i_ucPort	COM1	COM1

SIO_ModemDial

语法：

CHAR SIO_ModemDial(UCHAR i_ucPort, UCHAR * i_ucTelenum)

描述：

将 modem 连接到指定的电话号码。

参数 描述

I_ucPort 端口号

I_ucTelenum modem 将要连接到的电话号码。

返回值：

如果成功返回 0，如果错误返回值如下：

Illegal_Setting -5

备注：

参数	值	描述
i_ucPort	COM1	COM1

SIO_ModemHandup

语法：

CHAR SIO_ModemHandup(UCHAR i_ucPort)

描述：

设置 modem 响应电话接入

参数 描述

I_ucPort 端口号

返回值：

如果成功返回 0，如果错误返回值如下：

Illegal_Setting -5

备注：

参数	值	描述
i_ucPort	COM1	COM1

COM 口通讯示例 2（modem）：

```
#include "8187drv.h"
get_modem_response(char *buf)
{
   long i;
```

```
    int index;
    unsigned char c;
        index=0;
        for(i=0;i<10000; i++)
        {
          /*--- Get the receiving string from the Com1 port ---*/       if(Sio_recvbytes
                (Com1, unblock_mode,1,&c)>0) buf[index++]=c;
          if(index >0    && c == '\r')       /* end of command */
          {
            buf[index] =0;
            dsl_printf("Response : %s ",buf);
            return(1);
          }
        }
        return(0);
}
void main()
{
    char        c;
    int result_code;
    char        buf[131];
    int index;
    long        i;
    long        retry;

      SIO_ModemInitial(COM1);
  while(1)
  {
    dsl_printf("---------- Main Menu ---------\n");
    dsl_printf("0: Exit. \n");
    dsl_printf("1: COM port setting. \n");
    dsl_printf("2: Dial. \n");
    dsl_printf("3: Set to auto-answer. \n");
    dsl_printf("4: Set BREAK. \n");
    dsl_printf("5: Hand up. \n");
    dsl_printf("\n Please select a item to implement...\n");
    c=getch();
    switch(c)
    {
      case '0':
        return;
      case '1':
        /*--- Install the interrupt service routine for Com 1--*/ if(Sio_open(Com1)!=0)
        {
          printf("error\n");
          return;
        }
        if(Sio_setstate(Com1,(unsigned long)9600,no_parity,data8,stop1)!=0)
        {
          printf("Set State Error\n");
```

```
        return;
    }
    /*--- Show the data format on the screen ---*/
    dsl_printf("Com port is com1,baud rate is 9600 bps,data format is N,8,1\r\n");
    break;
case '2':
    /*--- Send prefix ---*/
    SIO_ModemCommand(COM1, "AT");
    /*--- Wait about 1 second--*/
    retry=100000;
    for(i=0;i<retry;i++)
    {
        i++;i--;
    }
    /*--- Clear the    buffers       of the transmitter and receiver       --*/
    SIO_PurgeBuf(COM1, Clear_RXBuffer);
    SIO_PurgeBuf(COM1, Clear_TXBuffer);
    /**--- Start to     dial ---*/
    /*--- Set DTR     is ON(1). --*/
    SIO_LowerRaise_RTS_DTR(COM1, RaiseSignal, Signal_DTR);
    /*--- Send the dialing command and phone number --*/
    sprintf(buf,"886222184867");
    SIO_ModemDial(COM1, buf); dsl_printf("Command    : %s \n",buf);
    /*--- Wait for the response from the other end --*/
    if( get_modem_response(buf)==1) dsl_printf("Response : %s \n",buf);
    else
        dsl_printf("Response : %s \n","No response");
    break;
case '3':
    /*--- After one    ring-bell, the phone is  answered automatically.       */
    SIO_LowerRaise_RTS_DTR(COM1, RaiseSignal, Signal_DTR);
    SIO_ModemAutoanswer(COM1);
    dsl_printf("Now is ready to get data...\n");
    break;
case '4':
    /* Set Break */
    SIO_SetBreak(COM1);
    dsl_printf("Now set a break to modem...\n\n\n");
    /*--- Wait about 0.3 second--*/
    retry=30000;
    for(i=0;i<retry;i++)
    {
        i++;i--;
    }
    SIO_ClearBreak(COM1);
    dsl_printf("Now clear the break ...\n\n\n");
    break;
case '5':
    /*--- Set DTR line OFF --*/
SIO_LowerRaise_RTS_DTR(COM1, LowerSignal, Signal_DTR);
```

```
            /*--- Wait about 0.3 second--*/
            retry=30000;
            for(i=0;i<retry;i++)
            {
               i++;i--;
            }
            /*--- Check whether DCD is off ---*/
            /* if(!(com_get_modem_status(0x3F8)&0x80)) break;*/
            /*--- Go to modem command state        ---*/
            SIO_ModemHandup(COM1);
            retry=3;
            do
            {
               SIO_ModemHandup(COM1);
               if(get_modem_response(buf)== 1)
                  {
                     if( buf[0] ==0)
                     break;
                  }
                  dsl_printf("retry %ld \n",4-retry);
            }
            while(--retry);
            dsl_printf("Now is hand up...\n");
            break;
         }
      }
   }
```

Ver_COMLib

语法：

void Ver_COMLib(char *vstr)

描述：

取得 COM 口函数库版本

参数	**描述**
vstr	指向 COM 口函数库版本号信息的指针

返回值：

无

示例：

```
char library_ver[20];
void main(void)
{
   Ver_COMLib(library_ver);
   dsl_printf("The version of library is %s\n", library_ver);
}
```

备注：

无

5.3.4 MODBUS/RTU 函数（RTU*.LIB）

在使用 Modbus 功能之前，请先阅读下面的快速入门：

怎样建立 Modbus TCP/RTU 服务器？

Modbus RTU 服务器：

步骤 1：Modbus_COM_Init(…)函数初始化 COM 口为 Modbus RTU 服务器模式。

步骤 2：RemoRTU_ModServer_Create(…)建立 Modbus RTU 服务器。

Modbus TCP 服务器：

步骤 1：使用 RemoTCP_ModServer_Create(…)函数建立 Modbus TCP 服务器。

步骤 2：调用 RemoTCP_ModServer_Update(…)函数定时检查是否有客户信息并保持服务器处于活动状态。

怎样建立 Modbus TCP/RTU 客户端？

Modbus RTU 客户端：

步骤 1：Modbus_COM_Init(…)函数初始化 COM 口为 Modbus RTU 客户端。

步骤 2：RemoModbusRTU_Read(…)函数接受来自服务器的数据，使用 RemoModbusRTU_Write(…)来发送数据到服务器上。

Modbus TCP 客户端：

步骤 1：使用 RemoTCP_Connect(…)建立到服务器的连接。

步骤 2：使用 RemoModbusTCP_Read(…)接收从服务器发来的数据，使用 RemoModbusTCP_Write(…)将数据发送到服务器。

示例：（Modbus RTU 服务器和 ModbusRTU 客户端）

```
#define SlaveID 1
int ModbusAddr_Mem[8]; //ModbusAddr_Mem[0]=>40001,...,
                       //ModbusAddr_Mem[7]=>40008
Modbus_Com_Init(Com1,Slave, (unsigned long)9600,no_parity,data8,stop1);
RemoRTU_ModServer_Create(SlaveID, (unsigned char*)ModbusAddr_Mem,
sizeof(ModbusAddr_Mem));
ModbusAddr_Mem[0]=0x1234; //40001=>0x1234
ModbusAddr_Mem[7]=0x4321; //40008=>0x4321
#define Read_StartAddr 40001 //query data start from address 40001
#define Read_EndAddr 40008 //query data end to address 40008
#define SlaveID 1
unsigned char Resp_From_Server[16];
//8 registers from 40001 to 40008.
//Therefore, we need at least a 16-byte physical memory.
int RespByteCount; //The total bytes of query data.
```

```
Modbus_Com_Init(Com1,Master,(unsigned long)9600,no_parity,data8,stop1)
RemoModbusRTU_Read(COM1, SlaveID, Read_StartAddr,
Read_EndAddr, &RespByteCount, Resp_From_Server);
Resp_From_Server[0] ==> 40001 Hi byte
Resp_From_Server[1] ==> 40001 Lo byte
```

示例：（Modbus TCP 服务器和 Modbus TCP 客户端）

```
#define TCP_Port 502 //502 is the standard port of modbus protocol
#define iTimeOut 3000 //3000 msec for timeout setting
#define iConns 20 //Only 20 client connections are avaliable
int ModbusAddr_Mem[8]; //ModbusAddr_Mem[0]=>40001,...,
                       //ModbusAddr_Mem[7]=>40008
RemoTCP_ModServer_Create(TCP_Port, iTimeOut, iConns, (unsigned char*)
  ModbusAddr_Mem, sizeof(ModbusAddr_Mem));
while(1)
//put REMOTCP_ModServer_Update() inside infinite loop
{   //for calling REMOTCP_ModServer_Update() periodically.
  if(RemoTCP_ModServer_Update()==HasMessage)
    //check if there is any client message
  {
    ...
  }
}
ModbusAddr_Mem[0]=0x1234; //40001=>0x1234
ModbusAddr_Mem[7]=0x4321; //40008=>0x4321
#define Server_Port 502 //502 is the standard port of modbus protocol
#define Server_IP "10.0.0.1" //the IP of server
#define iTimeOut 4000 //4000 msec for timeout setting
#define Read_StartAddr 40001 //query data start from address 40001
#define Read_EndAddr 40008 //query data end to address 40008
#define SlaveID 1
unsigned char Resp_From_Server[16];
//8 registers from 40001 to 40008. Therefore,
//we need at least a 16-byte physical memory.
int RespByteCount; //The total bytes of query data.
SOCKET SO;
REMOTCP_Connect(&SO, Server_IP, Server_Port);
REMOModbusTCP_Read(&SO, iTimeout, SlaveID, Read_StartAddr,
Read_EndAddr, &RespByteCount, Resp_From_Server);
Resp_From_Server[0] ==> 40001 Hi byte
Resp_From_Server[1] ==> 40001 Lo byte
```

Modbus_COM_Init

语法：

int Modbus_COM_Init(int Port, int iMode, unsigned long iBaud, int iParity, int iFormat, int iStopBits)

描述：

初始化一个 COM 口为 Modbus/RTU 连接。

参数	值	描述
Port	COM1	初始化 COM1
	COM2	初始化 COM2
	COM3	初始化 COM3
	COM4	初始化 COM4
IMode	Slave	Modbus/RTU 从模式
	Master	Modbus/RTU 主模式
IBaud	9600,etc…	波特率
IParity	NO_PARITY	无校验
	ODD_PARITY	奇校验
	EVEN_PARITY	偶校验
	ONE_PARITY	校验位为 1
	ZERO_PARITY	校验位为 0
IFormat	DATA5	5 个数据位
	DATA6	6 个数据位
	DATA7	7 个数据位
	DATA8	8 个数据位
IStopBits	STOP1	1 个停止位
	STOP2	2 个停止位

返回值：

0 没有错误

1 COM_already_installed：COM 口已经被安装

2 Err_Access_COM：当访问 COM 口的时候发生错误

示例：

```
if(Modbus_Com_Init(Com2,master,(unsigned long)9600,no_parrity,data8,stop1)!=0)
{
   dsl_printf("error\n");
   return;
}
dsl_printf("init success!!\n");
```

Modbus_COM_Release

语法：

void Modbus_COM_Release(int Port)

描述：

释放 COM 口的 Modbus 连接。

参数	值	描述
Port	1	COM1
	2	COM2
	3	COM3
	4	COM4

返回值：

无

Error_Code

语法：

int Error_Code(void)

描述：

当下列函数调用发生错误返回时，这个函数可以给客户提供准确的错误代码：

RemoRTU_ForceMultiCoils()
RemoRTU_ForceSingleCoil()
RemoRTU_PresetMultiRegs()
RemoRTU_PresetSingleReg()
RemoRTU_ReadCoilStatus()
RemoRTU_ReadHoldingRegs()
RemoRTU_ReadInputRegs()
RemoRTU_ReadInputStatus()

参数：

无

返回值：

空	没有错误产生
Erro Code	产生错误

错误代码：

91 无效的返回
92 COM 口初始化或模式错误
93 COM 口超时

示例：

参考 RemoRTU_ForceMultiCoils

RemoRTU_ForceMultiCoils

语法：

bool RemoRTU_ForceMultiCoils(int iPort,int Slave_Addr,intCoilIndex, int TotalPoint, int TotalByte, unsigned char szData[])

描述：

Modbus/RTU 的功能码“0F HEX”

参数	**描述**
iPort	COM 口号
Slave_Addr	从地址
CoilIndex	卷地址
TotalPoint	卷数量
TotalByte	字节数量
SzData[]	强制数据

返回值：

TRUE	没有错误产生
FALSE	产生了错误，调用 Error_Code()来获得具体错误代码

示例：

```
HostData[0]=0xf0;
if(!RemoRTU_ForceMultiCoils(COM1, 0x02, 0x64, 0x08, 0x01,HostData))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
}
else
dsl_printf("Success!!");
```

RemoRTU_ForceSingleCoil

语法：

bool RemoRTU_ForceSingleCoil(int iPort,int i_iAddr,int i_iCoilIndex,int i_iData)

描述：

Modbus/RTU 的功能码“05 HEX”

参数	**描述**
iPort	COM 口号
I_iAddr	从地址
I_iCoilIndex	卷地址
Int I_iData	强制数据

返回值：

TRUE	无错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

```
if(!RemoRTU_ForceSingleCoil(COM1, 0x02, 0x65, 0))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
}
else
dsl_printf("Success!!");
```

RemoRTU_PresetMultiRegs

语法：

bool RemoRTU_PresetMultiRegs(int iPort, int i_iAddr, int i_iStartReg, int i_iTotalReg, int i_iTotalByte, unsigned char i_szData[])

描述：

Modbus/RTU 功能代码“10 HEX”

参数	**描述**
iPort	COM 口号
I_iAddr	从地址
I_iStartReg	开始地址
I_iTotalReg	HI 寄存器号
I_iTotalByte	字节数
I_szData[]	数据

返回值：

TRUE	没有错误产生
FALSE	产生错误，调用 Error_Code()来获取错误代码

示例：

```
HostData[0]=0x12;
HostData[1]=0x56;
HostData[2]=0x38;
HostData[3]=0x09;
if(!RemoRTU_PresetMultiRegs(COM1, 0x02, 0x64, 2, 4, HostData))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
   return;
}
else
dsl_printf("Success!!");
```

RemoRTU_PresetSingleReg

语法：

bool RemoRTU_PresetSingleReg(int iport, int i_iaddr, int i_iregindex,int i_idata)

描述：

Modbus RTU 功能代码的“06 HEX”

参数	**描述**
iPort	COM 口号
I_iAddr	从地址
I_iRegIndex	寄存器地址
I_iData	预置数据

返回值：

TURE	没有错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

```
if(!RemoRTU_PresetSingleReg(COM1, 0x02, 0x68, 0x1234))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
   return;
}
else
dsl_printf("Success!!");
```

RemoModbusRTU_Write

语法：

bool RemoModbusTCP_Write(Socket*SO,int WaitMilliSec,intSlave_Addr, unsigned long i_iStartAddr, unsigned long i_iEndAddr,unsigned char i_szData[]);

描述：

预置值或强制卷在一个寄存器序列里，或一个卷序列里。

参数	**描述**
iPort	COM 口号
Slave_Addr	从设备的 ID 地址（从地址 ID 范围为 0-247）
I_iStartAddr	从设备的开始地址
I_iEndAddr	从设备的结束地址
I_szData	查询数据目录缓存

返回值：

如果成功 RemoModbusRTU_Write()返回 TRUE。如果错误，将产生一个错误代码，可以使用 Error_Code()来获取最后一个错误信息。

示例：

参考 Remo-8187\Source\Example\ModbusAppEx

RemoRTU_ReadCoilStatus

语法：

bool RemoRTU_ReadCoilStatus(int iPort,int i_iAddr,int i_iStartIndex,int i_iTotalPoint, int *o_iTotalByte, unsigned char o_szData[])

描述：

ModbusRTU 功能码“01 HEX”

参数	**描述**
iPort	COM 口号
I_iAddr	从地址
I_iStartIndex	开始地址
I_iTotalPoint	小数位数
O_iTotalByte	字节数
O_szData[]	卷数据

返回值：

TRUE	无错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

```
if(!RemoRTU_ReadCoilStatus(COM1,0x02,0x6E,0x01,&databytecount, hostdata))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
}
else
{
   dsl_printf("Status: ");
   for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
      {
         dsl_printf("%02X", HostData[tmpcnt]);
      }
      dsl_printf("\n");
}
```

RemoRTU_ReadHoldingRegs

语法：

bool RemoRTU_ReadHoldingRegs(int iport,int i_iaddr,int i_istartIndex,int i_itotalpoint, int *o_itotalbyte,unsigned charo_szdata[])

描述：

ModbusRTU 功能码“03 HEX”

参数	**描述**
iPort	COM 口号

I_iAddr	从地址
I_iStartIndex	开始地址
I_iTotalPoint	小数位数
O_iTotalByte	字节数
O_szData[]	寄存器数据

返回值：

TRUE	无错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

```
if(!RemoRTU_readholdingregs(Com1,0x02,0x65,0x01,&databytecount,hostdata))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
}
else
{
   dsl_printf("Status: ");
   for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
   {
      dsl_printf("%02X", HostData[tmpcnt]);
   }
   dsl_printf("\n");
}
```

RemoRTU_ReadinputRegs

语法：

bool RemoRTU_Readinputregs(int iport,int i_iaddr,int i_istartindex,inti_itotalpoint, int *o_iTotalByte, unsigned char o_szData[])

描述：

ModbusRTU 功能码“04 HEX”

参数	描述
iPort	COM 口号
I_iAddr	从地址
I_iStartIndex	开始地址
I_iTotalPoint	小数位数
O_iTotalByte	字节数
O_szData[]	寄存器数据

返回值：

TRUE	无错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

```
if(!RemoRTU_readinputregs(Com1,0x02,0x65,0x01,&databytecount,hostdata))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
}
else
{
   dsl_printf("Status: ");
   for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
   {
      dsl_printf("%02X", HostData[tmpcnt]);
   }
   dsl_printf("\n");
}
```

RemoRTU_ReadInputStatus

语法：

bool RemoRTU_ReadInputStatus(int iport,int i_iaddr,inti_istartindex,int i_itotalpoint, int *o_iTotalByte, unsigned charo_szData[])

描述：

ModbusRTU 功能码“02 HEX”

参数	**描述**
iPort	COM 口号
I_iAddr	从地址
I_iStartIndex	开始地址
I_iTotalPoint	小数位数
O_iTotalByte	字节数
O_szData[]	输入数据

返回值：

TRUE	无错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

```
if(!RemoRTU_ReadInputstatus(Com1,0x02,0x64,0x08,&databytecount,hostdata))
{
   dsl_printf("err code is %d\n", Error_Code());
   dsl_printf("fail send..");
}
else
{
   dsl_printf("Status: ");
```

```
    for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
    {
      dsl_printf("%02X", HostData[tmpcnt]);
    }
    dsl_printf("\n");
}
```

RemoModbusRTU_Read

语法：

bool RemoModbusRTU_Read(int iport,int slave_addr,unsigned long i_istartaddr, unsigned long i_iendaddr,unsigned int*o_ibyteofresp,unsigned char o_szresp[]);

描述：

读从设备的卷的参考开/关状态或寄存器的参考二进制目录。

参数	**描述**
iPort	COM 口号
Slave_Addr	从设备的地址 ID
I_iStartAddr	从设备的开始地址
I_iEndAddr	从设备的结束地址
I_iTotalPoint	要读取的卷/寄存器数量
O_iByteOfResp	返回的内容的总字节数
O_szResp	返回内容缓存

返回值：

TRUE	无错误产生
FALSE	产生错误，调用 Error_Code()来获得错误代码

示例：

参考 RemoDAQ-8187\Source\Example\ModbusAppEx

RemoRTU_ModServer_Create

语法：

void RemoRTU_ModServer_Create(int slave_addr,unsigned char *ptr_mem, unsigned int size_of_mem)

描述：

建立 Modbus/RTU 服务器

参数	**描述**
slave_addr	Modbus/RTU 服务器的从地址
ptr_mem	共享存储器
size_of_mem	共享存储器的大小

返回值：

无

示例：

```
RemoRTU_ModServer_Create(3, (unsigned char *)Share_Mem,sizeof(Share_Mem));
dsl_printf("server started..\n");
while(1)
{
   if(predate != Share_Mem[0])
   {
      dsl_printf("40001 is %X\n", Share_Mem[0]);
      //strongly recommend use dsl_printf() instead of
      printf()
      predate = Share_Mem[0];
   }
}
```

Ver_RTU_Mod

语法：

void Ver_RTU_Mod(char *vstr)

描述：

检查 Modbus/RTU 函数库的版本

参数	**描述**
vstr	Modbus/RTU 函数库的版本信息指针

返回值：

无

示例：

```
char library_ver[20];
void main(void)
{
   Ver_RTU_Mod(library_ver);
   dsl_printf("The version of library is %s\n", library_ver);
}
```

5.3.5 MODBUS/TCP 功能（MBTCP*.LIB）

Ver_TCP_Mod

语法：

void Ver_TCP_Mod(char *vstr)

描述：

检查 Modbus/TCP 函数库的版本

参数	**描述**
vstr	Modbus/TCP 函数库的版本信息指针

返回值：

无

示例：

```
char library_ver[20];
void main(void)
{
   Ver_TCP_Mod(library_ver);
   dsl_printf("The version of library is %s\n", library_ver);
}
```

Modbus TCP 客户端功能：

ReturnErr_code

语法：

int ReturnErr_code(void)

描述：

如果下列函数发生错误，则这个函数可以获得错误代码：

RemoTCP_ForceMultiCoils()
RemoTCP_ForceSingleCoil()
RemoTCP_PresetMultiRegs()
RemoTCP_PresetSingleReg()
RemoTCP_ReadCoilStatus()
RemoTCP_ReadHoldingRegs()
RemoTCP_ReadInputRegs()
RemoTCP_ReadInputStatus()

参数：无

返回值：

NULL	没有错误产生
Erro Code	错误代码

错误代码：

01	非法函数	02	非法数据地址
03	非法数据	04	从设备错误

05 确认　　06 从设备忙
07 拒绝确认　　08 内存奇偶错误

示例：

```
SOCKET SO_8187;
…
if(RemoTcp_readcoilstatus(&so_8187,50,0x01,0x11,0x10,&databytecount,hostdata)<=0)
{
  perror("RemoTCP_ReadCoilStatus()\n");
  dsl_printf("err code is %d\n", ReturnErr_code());
  RemoTCP_Disconnect(&SO_8187);
  return 0;
}
```

RemoTCP_Connect

语法：

int RemoTCP_Connect(SOCKET * SO, char * Target_IP, intTarget_Port)

描述：

连接到 Modbus/TCP 服务器

参数	**描述**
SO	一个未连接的套接字标识符
Target_IP	Modbus/TCP 服务器 IP
Target_Port	连接服务器的端口

返回值：

TRUE	没有错误产生
-1	当获取主机名时产生错误
-2	要初始化的套接字是非法的
-3	当连接 Modbus/TCP 服务器时产生错误

示例：

```
if(RemoTCP_Connect(&SO_8187, ServerIP, Server_Port)<=0)
{
  perror("REMOTCP_Connect()\n");
  RemoTCP_Disconnect(&SO_8187);
  return 0;
}
```

RemoTCP_Disconnect

语法：

bool REMOTCP_Disconnect(SOCKET * SO)

描述：断开 Modbus/TCP 服务器的连接

参数：

SO 识别已连接到 Modbus/TCP 服务器的套接字的标识符

返回值：

TRUE 没有错误产生

FALSE 产生了错误

示例：

```
if(RemoTCP_Connect(&SO_8187, ServerIP, Server_Port)<=0)
{
  perror("REMOTCP_Connect()\n");
  RemoTCP_Disconnect(&SO_8187);
  return 0;
}
```

RemoTCP_ForceMultiCoils

语法：

int RemoTCP_ForceMultiCoils(socket*so,int waitmillisec,intslave_addr,int coilIndex, int totalpoint,int totalbyte,unsigned charszdata[])

描述：

Modbus TCP 功能码：“0F HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
CoilIndex	卷地址
TotalPoint	卷数量
TotalByte	字节数
SzData[]	强制数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
HostData[1]=~0x33;//force channel status to 0x3333
if(RemoTCP_Forcemulticoils(&so_8187,50,0x01,0x21,0x10,0x02,hostdata)<=0)
{
  perror("REMOTCP_ForceMultiCoils()\n");
  RemoTCP_Disconnect(&SO_8187);
  return 0;
}
```

RemoTCP_ForceSingleCoil

语法：

int RemoTCP_ForceSingleCoil(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int CoilIndex, int Data)

描述：

Modbus TCP 功能码：“05 HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
CoilIndex	卷地址
Data	强制数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
if(RemoTCP_ForceSingleCoil(&SO_8187, 50, 0x01, 0x25, 1)<=0)
{
   perror("RemoTCP_ForceSingleCoil()\n");
   RemoTCP_Disconnect(&SO_8187);
   return 0;
}
```

RemoTCP_PresetMultiRegs

语法：

int RemoTCP_PresetMultiRegs(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int StartReg, int TotalReg, int TotalByte, unsigned charData[])

描述：

Modbus TCP 功能码：“10 HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
StartReg	开始地址
TotalReg	寄存器数量
TotalByte	字节数
SzData[]	数据

返回值：

TRUE	没有错误
0	超时错误

-1	发送数据时错误
-2	接收数据时错误

示例：

```
HostData[0]=0x07;
HostData[1]=0x00;
HostData[2]=0x07;
HostData[3]=0x00;
if(RemoTCP_PresetMultiRegs(&SO_8187,50,0x01,0x19,0x02,4,HostData)<=0)
{
   perror("RemoTCP_PresetMultiRegs()\n");
   RemoTCP_Disconnect(&SO_8187);
   return 0;
}
```

RemoTCP_PresetSingleReg

语法：

int RemoTCP_PresetSingleReg(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int RegIndex, int Data)

描述：

Modbus TCP 功能码：“06 HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
RegIndex	寄存器地址
Data	初始数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
if(RemoTCP_PresetSingleReg(&SO_8187, 50, 0x01, 0x19,0x07ff)<=0)
{
   perror("RemoTCP_PresetSingleReg()\n");
   RemoTCP_Disconnect(&SO_8187);
   return 0;
}
```

RemoModbusTCP_Write

语法：

bool RemoModbusTCP_Write(SOCKET * SO, int WaitMilliSec, int Slave_Addr,

unsigned long i_iStartAddr, unsigned long i_iEndAddr,unsigned char i_szData[]);

描述：

初始化寄存器值，或卷值

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	REMOModbus TCP_Write()的最大时间
Slave_Addr	从设备的 ID 地址
I_iStartAddr	从设备的开始地址
I_iEndAddr	从设备的结束地址
I_szData	查询数据目录的缓存

返回值：

如果 RemoModbus TCP_Write()成功则返回 TRUE。如果错误，则返回错误 FALSE，可以用 Error_Code()来获得最后一个错误代码

示例：

参考：RemoDAQ-8187\Source\Example\ModbusAppEx 下的程序

RemoTCP_ReadCoilStatus

语法：

int RemoTCP_ReadCoilStatus(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int StartIndex, int TotalPoint, int * ByteCount, char *wData)

描述：

Modbus TCP 功能码：“01 HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
StartIndex	开始地址
TotalPoint	指针数量
ByteCount	字节数量
wData	数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
if(Remotcp_readcoilstatus(&so_8187,50,0x01,0x11,0x10,&databytecount,hostdata)<=0)
{
   perror("RemoTCP_ReadCoilStatus()\n");
```

```
    RemoTCP_Disconnect(&SO_8187);
    return 0;
  }
  else
    { dsl_printf("Remo-8187 Status: ");
      for(tmp=0; tmp<DataByteCount; tmp++)
      {
        dsl_printf("%2X", HostData[tmp]);
      }
    dsl_printf("\n");
  }
```

RemoTCP_ReadHoldingRegs

语法：

int RemoTCP_ReadHoldingRegs(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int StartIndex, int TotalPoint, int * ByteCount, char *wData)

描述：

Modbus TCP 功能码：“03 HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
StartIndex	开始地址
TotalPoint	指针数量
ByteCount	字节数量
wData	数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
if((errno=RemoTCP_readholdingregs(&so_8187,50,0x01,0x19,0x08,&databytecount,
  HostData))<=0)
  {
    perror("RemoTCP_ReadHoldingRegs()\n");
    dsl_printf("errno is %d\n", errno);
    RemoTCP_Disconnect(&SO_8187);
    return 0;
  }
  else
  {
    dsl_printf("REMO-5024 Status: ");
```

```
    for(tmp=0; tmp<DataByteCount; tmp++)
    {
      dsl_printf("%02X", HostData[tmp]);
    }
    dsl_printf("\n");
  }
```

RemoTCP_ReadInputRegs

语法：

int RemoTCP_ReadInputRegs(Socket*so,int WaitMilliSec,intSlave_Addr,int StartIndex, int TotalPoint, int * ByteCount, char *wData)

描述：

Modbus TCP 功能码：“04 HEX”

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
StartIndex	开始地址
TotalPoint	指针数量
ByteCount	字节数量
wData	数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
if((errno=RemoTCP_ReadInputRegs(&SO_8187,50,0x01,0x19,0x08,DataByteCount,
  HostData))<=0)
  {
    perror("RemoTCP_ReadInputRegs()\n");
    dsl_printf("errno is %d\n", errno);
    RemoTCP_Disconnect(&SO_8187);
    return 0;
  }
  else
  {
    dsl_printf("REMO-5024 Status: ");
    for(tmp=0; tmp<DataByteCount; tmp++)
    {
      dsl_printf("%02X", HostData[tmp]);
    }
    dsl_printf("\n");
```

```
}
```

RemoTCP_ReadInputStatus

语法：

int RemoTCP_ReadInputStatus(socket*so,int waitmillisec,intSlave_Addr,int StartIndex, int TotalPoint, int * ByteCount, char *wData)

描述：

Modbus TCP 功能码：“02 HEX”

参数	描述
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	设置从 Modbus/TCP 服务器返回数据的等待时间（毫秒级）
Slave_Addr	从地址
StartIndex	开始地址
TotalPoint	指针数量
ByteCount	字节数量
wData	数据

返回值：

TRUE	没有错误
0	超时错误
-1	发送数据时错误
-2	接收数据时错误

示例：

```
if(REMOTCP_ReadInputStatus(&SO_8187,50,0x01,0x11,0x10,&DataByteCount,
    HostData)<=0)
  {
    perror("REMOTCP_ReadInputStatus()\n");
    REMOTCP_Disconnect(&SO_8187);
    return 0;
  }
  else
    {
      dsl_printf("REMO-5051 Status: ");
      for(tmp=0; tmp<DataByteCount; tmp++)
      {
        dsl_printf("%2X", HostData[tmp]);
      }
    dsl_printf("\n");
    }
```

REMOModbusTCP_Read

语法：

bool REMOModbusTCP_Read(socket*so,int waitmillisec,intslave_Addr,unsigned long

i_istartaddr,unsigned long i_iendaddr,unsigned int*o_ibyteofresp,unsigned char o_szresp[]);

描述：

读取卷索引的开关状态或从设备的寄存器索引的二进制目录。

参数	**描述**
SO	连接到 Modbus/TCP 服务器的套接字
WaitMilliSec	REMOModbus TCP_Write()的最大时间
Slave_Addr	从设备的 ID 地址
I_iStartAddr	从设备的开始地址
I_iEndAddr	从设备的结束地址
I_iTotalPoint	要读取的卷或寄存器的数量
O_iByteOfResp	返回数据内容的总字节数
O_szResp	返回数据内容缓存

返回值：

如果 REMOModbus TCP_Write()成功则返回 TRUE。如果错误，则返回错误 FALSE，可以用 Error_Code()来获得最后一个错误代码

示例：

参考 RemoDAQ-8187\Source\Example\ModbusAppEx 下的示例

Modbus TCP 服务器函数：

REMOTCP_ModServer_Create

语法：

int REMOTCP_ModServer_Create(int Host_Port,unsigned longwaittimeout,unsigned int numberConns, unsigned char * ptr_mem, intsize_mem)

描述：

建立 Modbus/TCP 服务器

参数	**描述**
Host_Port	Modbus/TCP 服务器的断口
Waittimeout	超时值，0-0xffffffff 毫秒级
NumberConns	客户端最大连接数
Prt_mem	共享存储器
Size_mem	共享存储器大小

返回值：

0 没有错误
91 非法套接字
92 当关联一个本地套接字时发生错误
93 当设置套接字时发生错误
94 当监听输入套接字时发生错误

示例：

```
if((err_code=REMOTCP_modserver_create(502,5000,20,(unsigned char *) share_mem,
sizeof(Share_Mem)))!=0)
{
   dsl_printf("error code is %d/n", err_code);
}
dsl_printf("Server started, wait for connect...\n");
```

REMOTCP_ModServer_Update

语法：

int REMOTCP_ModServer_Update(void)

描述：

更新 Modbus/TCP 服务器。Modbus/TCP 服务器需要通过不断调用 RemoTCP_ ModServer_Update()函数来更新服务器，使其保持活动状态。

参数：

无

返回值：

1 有新消息

0 没有新消息

示例：

```
while(1)
{
   iState = REMOTCP_ModServer_Update(); //second step
   if(iState) //if has message, show the data at address 40001
   {
      if(pre_data != Share_Mem[0])
      {
         dsl_printf("40001 is %X\n", Share_Mem[0]);
         //notice: printf() will decrease server performance
         pre_data = Share_Mem[0];
      }
   }
}
```

REMOTCP_ModServer_Release

语法：

void REMOTCP_ModServer_Release(void)

描述：

释放 Modbus/TCP 服务器

参数：

无

返回值：

无

5.3.6 套接字函数（SOCKET*.LIB）

Accept

语法：

SOCKET accept (SOCKET so, struct sockaddr *psAddress, int *piAddressLen)

描述：

在套接字上应答一个连接

参数：

so	在 listen()函数之后监听连接的套接字描述符
psAddress	一个可选的缓存指针
piAddrLen	一个可选的十进制指针，包含 psAddress 的地址

返回值：

如果没有错误，accept()返回公认的数据包描述符的套接字类型值。否则，将返回一个非法套接字值，并且一个错误代码将返回到 errno。

错误代码：

ENETDOWN	网络子系统失败
EFAULT	变量 piAddressLen 太小
EINVAL	listen()没有预先调用 accept()
EMFILE	在进入 accept()时队列为空，并且没有可用的描述符
ENOBUFS	没有可用的缓存空间
ENOTSOCK	描述符不是一个套接字
EOPNOTSUPP	参考套接字不支持定向连接服务
EWOULDBLOCK	套接字被标记为无分组盒无当前连接

备注：

相关函数：bind(),connect(),listen(),select(),socket()

Bind

语法：

int bind (SOCKET so, const struct sockaddr * psAddress, intiAddressLen)

描述：

将一个套接字和一个本地套接字地址相关联

参数：

so	自由套接字识别标识符

psAddress	分配一个套接字地址给套接字。Sockaddr 结构如下：
	struct sockaddr
	{
	u_short sa_family;
	char sa_data[14];
	};

iAddressLen　　psAddress 的名字长度

返回值：

如果没有错误，bind()返回 0。否则返回 SOCKET_ERROR，一个指定的错误代码送入 errno.

错误代码：

ENETDOWN	SOCKETS 检测到网络子系统出错
EADDRINUSE	指定的地址已经被使用
EFAULT	变量 iAddressLen 太短
EAFNOSUPPORT	协议不支持指定的地址
EINVAL	套接字已被限制为一个地址
ENOBUFS	没有足够的缓存，连接太多
ENOTSOCK	描述符不是一个套接字

备注：

相关函数：connect(),listen(),getsockname(),setsockopt(),socket()

Closesocket

语法：

int closesocket(SOCKET so)

描述：

关闭一个套接字

参数	**描述**
so	识别套接字的标识符

返回值：

如果没有错误，返回 0。否则返回 SOCKET_ERROR 值，错误代码写入 errno。

错误代码：

ENETDOWN	SOCKETS 检测到子网络错误
ENOTSOCK	描述符不是一个套接字
EWOULDBLOCK	套接字被设置为非阻塞,SO_LINGER 超时值被设置为非 0

备注：

相关函数：accept(),ioctlsocket(),setsockopt

Connect

语法：

int connect (SOCKET so, const struct sockaddr * psAddress, intiAddressLen)

描述：

在同一层建立一个连接

参数	描述
so	识别无连接套接字的描述符
psAddress	被连接的套接字在同一层的地址
iAddressLen	psAddress 长度

返回值：

如果没有错误，返回 0。否则返回 SOCKET_ERROR，并且错误代码被写入 errno。

错误代码：

ENETDOWN	套接字检测到网络子系统错误
EADDRINUSE	指定的地址已经被使用了
EADDRNOTAVAIL	指定的本机地址不可用
EAFNOSUPPORT	指定的地址段不能被用于这个套接字
ECONNREFUSED	请求连接被拒绝
EDESTADDREQ	需要一个目的地址
EFAULT	变量 iAddressLen 错误
EINVAL	套接字没有被指定一个地址
EISCONN	套接字已经被连接
ENETUNREACH	当时此主机的网络不通
ENOBUFS	没有可用的缓存空间，套接字不能被连接
ENOTSOCK	描述符不是一个套接字
ETIMEDOUT	尝试建立连接时超时，没有确认连接
EWOULDBLOCK	套接字被标记为无连接，并且连接不能被立即完成

备注：

这个函数用来建立一个指定的外部套接字地址。

相关函数：accept(),bind(),getsockname(),socket(),select()

Getpeername

语法：

int getpeername (SOCKET so, struct sockaddr * psAddress, int *piAddressLen)

描述：

取得已连接套接字同一层的套接字地址

参数	描述
so	一个识别已连接套接字的标识符
psAddress	取得同层套接字地址的数据结构
piAddressLen	psAddress 大小的指针结构

返回值：

如果成功返回 0。否则返回 SOCKET_ERROR，并且一个指定的错误代码写入 errno。

错误代码：

ENETDOWN	套接字检测到网络子系统错误

EFAULT	piAddressLen 变量长度不够
ENOTCONN	套接字未连接
ENOTSOCK	标识符不是一个套接字

备注：

相关函数：bind(),socket(),getsockname()。

Getsockname

语法：

int getsockname (SOCKET so, struct sockaddr * psAddress, int *piAddressLen)

描述：

为一个套接字获取本地套接字

参数	**描述**
so	识别套接字范围的标识符
psAddress	取得套接字的套接字地址
piAddressLen	一个指向 psAddress 缓存大小的指针

返回值：

如果成功返回 0。否则返回 SOCKET_ERROR，并且一个指定的错误代码写入 errno。

错误代码：

ENETDOWN	套接字检测到网络子系统错误
EFAULT	piAddressLen 变量长度不够
ENOTSOCK	标识符不是一个套接字
EINVAL	套接字和 bind()不是一个地址

备注：

相关函数：bind(),socket(),getpeername()。

Getsockopt

语法：

int getsockopt (SOCKET so, int iLevel, int iOptname, char * pcOptval,int * piOptlen)

描述：

取得套接字选项

参数	**描述**
so	套接字标识符
iLevel	选项定义的层数。
IOptname	套接字选项的返回值
PcOptval	指向请求项返回值的指针
PiOptlen	指向 pcOptval 缓存大小的指针

返回值：

如果成功返回 0。否则返回 SOCKET_ERROR，并且一个指定的错误代码写入 errno。

错误代码：

ENETDOWN	套接字检测到网络子系统错误
EFAULT	piOptlen 变量非法
ENOPROTOOPT	选项无法识别或不支持
ENOTSOCK	标识符不是一个套接字

备注：

相关函数：setsockopt(),socket()。

Htonl

语法：

u_long htonl (u_long *ulHostlong*)

描述：

将 u_long 从主机转换为网络字节格式

参数	**描述**
ulHostlong	一个主机字节格式的 32 位数

返回值：

htonl()返回网络字节格式的值

备注：

相关函数：htons(),ntohl(),ntohs()

Htons

语法：

u_short htons (u_short usH*ostshort*)

描述：

将 u_short 从主机模式转换为网络字节格式

参数	**描述**
sHostshort	一个主机字节格式的 16 位数

返回值：

返回网络字节格式的值

备注：

相关函数：htonl(),ntohl(),ntohs()

Inet_addr

语法：

unsigned long inet_addr (const char * *pc*)

描述：

将一个字符串格式的带小数点的地址转换为 in_addr 格式

参数	**描述**
pc	一个含有“.”的标准的以太网地址

返回值：

如果成功,返回这个给定网络地址的无符号二进制数.如果错误返回 inaddr_none 值

备注：

相关函数：inet_ntoa()

inet_ntoa

语法：

char * inet_ntoa (struct in_addr *sIn*)

描述：

将一个网络地址转换为一串带小数点格式的数据

参数	**描述**
sln	表示一个以太网主机地址的结构体

返回值：

如果成功返回一个包含小数点的标准 IP 地址格式。否则返回 NULL。

备注：

相关函数：inet_addr()

Ioctlsocket

语法：

int ioctlsocket (SOCKET so, long lCmd, u_long * pulArgp)

描述：

控制套接字的模式

参数	**描述**
so	一个识别套接字的标识符
iCmd	在套接字上执行的命令
pulArgp	指向 Icmd 参数的指针

返回值：

如果成功，返回 0。否则返回 SOCKET_ERROR 值,一个指定的错误代码被写入 errno

错误代码：

ENETDOWN	套接字检测到网络子系统错误

EINVAL	Icmd 为一个非法的命令
ENOTSOCK	标识符不是一个套接字

备注：

相关函数：socket(),setsockopt(),getsockopt()。

Listen

语法：

int listen (SOCKET *so*, int *iBacklog*)

描述：

建立一个套接字来监听接入的连接。

参数	**描述**
so	识别一个受限制的，没有连接的套接字的标识符
iBacklog	挂起的连接队列可能增长的最大长度

返回值：

如果没有错误产生返回 0。否则返回 SOCKET_ERROR，并且一个指定的错误代码被写入 errno。

错误代码：

ENETDOWN	套接字检测到网络子系统错误
EADDRINUSE	试图去监听一个正在使用的地址
EINVAL	套接字和 bind()不在一个范围，或者套接字已经连接
EISCONN	套接字已经连接
ENOBUFS	无可用缓存空间
ENOTSOCK	标识符不是一个套接字
EOPNOTSUPP	引用的套接字不支持 listen()操作

备注：

相关函数：accept(),connect(),socket()。

Ntohl

语法：

u_long ntohl (u_long *ulNetlong*)

描述：

将 u_long 从网络格式转换为主机格式

参数	**描述**
ulNetlong	一个网络格式的 32 位数

返回值：

返回主机格式

备注：

相关函数：htonl(),htons(),ntohs()。

Ntohs

语法：

u_short ntohs (u_short *usNetshort*)

描述：

将 u_short 从网络类型转换为主机字节类型.

返回值：

返回主机字节类型

参数	**描述**
usNetshort	一个网络字节类型的 16 位数

备注：

相关函数：htonl(),htons(),ntohl()

Recv

语法：

int recv (SOCKET so, char * pcbuf, int iLen, int iFlags)

描述：

从套接字返回值

返回值：

如果成功返回接收的字节数，如果错误返回 0，和错误代码。

Recvfrom

语法：

int recvfrom (SOCKET so, char * pcBuf, int iLen, int iFlags, struct sockaddr * psFrom, int * piFromlen)

描述：

接收数据并存储源地址

返回值：

如果成功返回接收的字节数。如果错误返回 0 和错误代码。

Select

语法：

int select (int iNfds, fd_set * psReadfds, fd_set * psWritefds, fd_set *psExceptfds, const struct timeval * psTimeout)

描述：

确定一个或多个套接字的状态。

返回值：

返回以准备好的套接字数量。如果错误返回错误代码。

Send

语法：

int send (SOCKET so, const char * pcBuf, int iLen, int iFlags)

描述：

将数据发送到一个以连接的套接字上。

返回值：

如果成功返回发送的字节数。否则返回错误代码。

Sendto

语法：

int sendto (SOCKET so, const char * pcBuf, int iLen, int iFlags, conststruct sockaddr * psTo, int iTolen)

描述：

将数据发送到一个特殊的地址

返回值：

如果成功返回发送的字节总数。否则返回错误代码。

Setsockopt

语法：

int setsockopt (SOCKET so, int level, int optname, const char * optval,int optlen)

描述：

设置套接字

返回值：

如果成功返回 0。否则返回错误代码。

Shutdown

语法：

int shutdown (SOCKET so, int how)

描述：

禁止套接字的发送和接收。

返回值：

如果成功返回 0。否则返回错误代码。

Socket

语法：

SOCKET socket (int af, int type, int protocol)

描述：

创建一个套接字。

返回值：

如果成功返回新套接字的描述符。否则，返回错误代码。

Gethostbyaddr

语法：

struct hostent * gethostbyaddr (const char * pcAddr, int len, int type)

描述：

获得对应地址的主机信息。

返回值：

如果成功，返回返回描述主机信息的指针。否则返回错误代码。

Gethostbyname

语法：

struct hostent * gethostbyname (const char * *pszName*)

描述：

获得对应一个主机名称的主机信息。

返回值：

如果成功，返回对应主机信息的指针。否则，返回错误代码。

Gethostname

语法：

int gethostname (char * pszName, int iAddressLen)

描述：

返回本地主机的标准主机名称。

返回值：

如果成功，返回 0。否则返回错误代码。

Getprotobyname

语法：

struct protoent * getprotobyname (const char * pszName)

描述：

获得对应协议名称的协议信息。

返回值：

如果成功，返回对应信息的指针。否则返回错误代码。

Getprotobynumber

语法：

struct protoent * getprotobynumber (int number)

描述：

获得对应协议号的协议信息。

返回值：

如果成功，返回对应信息的指针。否则返回错误代码。

Getservbyname

语法：

struct servent * getservbyname (const char * pszName, const char *proto)

描述：

获得对应服务器的名称和协议信息。

返回值：

如果成功，返回指向返回信息的指针。否则返回错误代码。

Getservbyport

语法：

struct servent * getservbyport (int port, const char * proto)

描述：

获得对应端口号和协议的信息。

返回值：

如果成功返回指向返回信息的指针。否则返回错误代码。