DCON Utility (DOS)

User Manual

7000 New Features
1. Self Tuner Inside
2. Multiple Baud Rate
3. Multiple Data Format
4. Dual WatchDog Inside
5. True Distributed Control
6. High Speed&High Density I/O

→ **Your Powerful Tools**

↓

Create New Ideas

↓

Create New Applications

Table of Contents

1 Introduction

The DCON Utility (DOS) provides utility, demo programs and data acquisition subroutines for the user to handle 7000/8000/87000 series modules. The 7000/8000/87000 are a family of remote controllable data acquisition modules. They provide A/D, D/A, DI/O, Timer/Counter, MMI and other functions. The 7000 modules operate with a simple command / response protocol to control all module functions.

1.1 Directory Contents

The contents of the directory is given as following:

\README.txt	→ read me file
\DCON_Utility_DOS\DIAG\*.*	→ diagnostic utility
\DCON_Utility_DOS \learning*.*	→ learning kits utility
\DCON_Utility_DOS \PC_NET*.*	→ PC-networking application utility
\DCON_Utility_DOS \PLC_NET*.*	→ 7000 PLC-networking application utility

1.2 Compiler & Link Using TC

- The including file → **UART.H**
- There are 2 source files → **UART.C & TEST.C**
- The project file → **TEST.PRJ**
- Use TC integrated environment to **select large model & project file**
- press F9 to compiler & link

1.3 Q&A

(Q1) How to program the 7000/8000/87000 modules ?

(A1) The user can use any RS-232C device to send the command string to the 7000/8000/87000 modules and the 7000/8000/87000 modules will echo the result string to the RS-232C device. The data format of command character is given as following :

Data format=1_start+8_data+no_parity+1_stop
total=1+8+1=10 bits per character

The user shouldn't change this format to the following invalidate format:

1_start+8_data+no_parity+2_stop → invalidate format

1_start+8_data+even_parity+1_stop → invalidate format

1_start+8_data+odd_parity+1_stop → invalidate format

1_start+7_data+no_parity+2_stop → invalidate format

1_start+7_data+even_parity+1_stop → invalidate format

1_start+7_data+odd_parity+1_stop → invalidate format

(Q2) What programming language can be used ?

(A2) Any programming language can be used to program the 7000/8000/87000 modules, if this language can send out the RS-232C command string. It is recommended to use the C/C++ , VB, and Delphi.

(Q3) Can the baud rate be changed?

(A3) Yes, the validate baud rate are 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200 BPS. PC can send out the command string under all these baud rate. Refer to Sec. 2.1.1 for baud rate change.

(Q4) Can the checksum of the command string be enabled?

(A4) Yes, but the default condition of checksum is DISABLE. Refer to Sec. 2.1.1 for enable/disable checksum. To ENABLE the checksum, two more checksum char must be added to the original command string. So the command string in checksum enable mode is two more bytes longer than the default checksum disable command string.

$012[0x0D]	→ command string under checksum DISABLE
$012B1[0x0D]	→ command string under checksum ENABLE

(Q5) How to calculate the checksum ?

(A5) The steps to calculate the checksum are given as following:

1. step 1: checksum=0
2. step 2: for all command byte checksum = checksum + command byte
3. step 3: checksum=checksum&0xff
4. step 4 : convert checksum to ASCII high byte and ASCII low byte

for example,

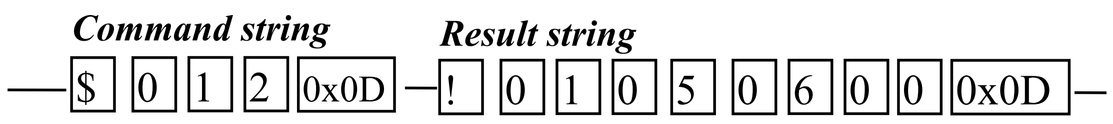
command = **$012[Enter]**
checksum = **$+0+1+2 =** 0x24+0x30+0x31+0x32
= 0xB7
checksum & 0xff = 0xB7
checksum ASCII high byte = ASCII B = 0x42
checksum ASCII low byte = ASCII 7 = 0x37
command with checksum = **$012B7[Enter]**

(Q6) What’s the syntax of the command string ?

(A6) The syntax of command string is shown as following:

[leading char][address][data][checksum][0x0D]

[leading char]	= **$** or **%** or **#** or **@** or **~**
[address]	= 00 to FF → 256 different module address max.
[data]	= command code and the related data if needed
[checksum]	= two chars if checksum enable
	= no char if checksum disable
[0x0D]	= ASCII code 0x0d, the command string terminator



(Q7) Can I use the 7000/8000/87000 module in the RS-232C network ?

(A7) No, the 7000/8000/87000 use the RS-485 network. So the users need a 7520, RS-232 to RS-485 converter, to convert the RS-232C signal to the RS-485 signal. That is to say that the users need at least two modules to form a minimum 7000/8000/87000 system. One is the 7520 and another is 7000/8000/87000 module (A/D or D/A or DI/O or Timer/Counter modules).

(Q8) Can the 7000/8000/87000 be used as a full duplex RS-485 system ?

(A8) NO, the 7000/8000/87000 is worked in a half duplex RS-485 system. The 7000/8000/87000 use a two-wire, D+ and D-, differential RS-485 network. The D+ and D- signal logically form a single bi-direction data line, so this single data line can send or receive data in the different time. Therefore the 7000/8000/87000 can't send the command string and receive the result string at the same time. The host computer must send the command string first then wait for the result string returned by the 7000/8000/87000.

(Q9) Who handle the direction of two-wire RS-485 network ?

(A9) The two-wire RS-485 network is a pairs of differential signal. So logically these two line equals to one data line. This data line can be used to transfer command and receive result both. This needs a direction controller to handle the two-wire RS-485 network. The 7520, RS-232 to RS-485 converter, is a intelligent controller. It will automatically detect and control the direction of data flow. When the host computer send RS-232C command string, the 7520 set the direction of the RS-485 network in received mode and the 7000/8000/87000 modules can receive the host's command string. After the command string is finished, the 7520 will set the RS-485 network in send mode to allow the 7000/8000/87000 modules to send the result to the host computer. Everything is automatically done, the user need not to care about the direction of data flow. The user only has to care that he should send the command string first then waiting for the result string next. The RTS-control signal of RS-232 is ignored. The 7000/8000/87000 don't use the RTS signal.

(Q10) Can I use the 7000/8000/87000 in the conventional 4-wire RS-485 network ?

(A10) No. The 7000/8000/87000 is designed for 2-wire RS-485 network

(Q11) How can the 7000/8000/87000 modules identify the correct command ?

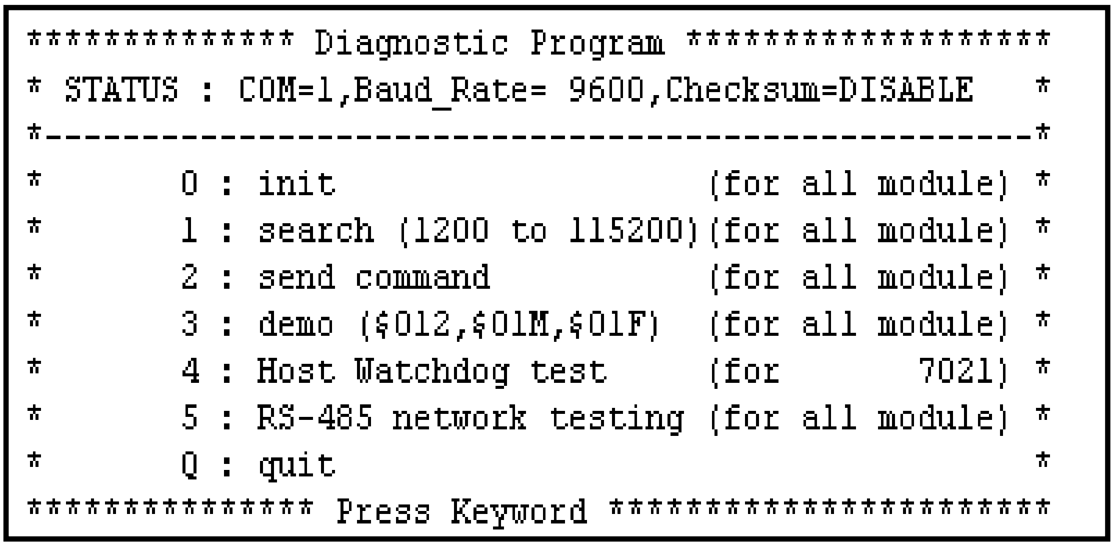
(A11) Every **7000/8000/87000** module has it's own module address. The command string always include the desired module's address. When the host computer send the command string, all the 7000 modules in the RS-485 network will receive this command string at the same time. Then All the 7000/8000/87000 modules will interpret this command string simultaneously. They will fetch the address field and identify the destination address. Only the module with the same address as the destination address will response to the host computer. All the other module in the network will bypass this command string and waiting for the next command.

For example, $012[0x0D] is a validate command string and the destination address is 01. The 7000/8000/87000 module with address 01 will response to this command.

2 Diagnostic Program

2.1 Functions Description

The diagnostic program \DCON_Utility_DOS\DIAG\TC\TEST.EXE can be executed in the environment of DOS, Windows 3.1, Windows 95, Windows NT. Run TEST.EXE, the screen will be shown as following:



```
*************** Diagnostic Program ********************
* STATUS : COM=1,Baud_Rate= 9600,Checksum=DISABLE     *
*-----------------------------------------------------*
*       0 : init                   (for all module) *
*       1 : search (1200 to 115200)(for all module) *
*       2 : send command           (for all module) *
*       3 : demo ($012,$01M,$01F)  (for all module) *
*       4 : Host Watchdog test     (for       7021) *
*       5 : RS-485 network testing (for all module) *
*       Q : quit                                     *
*************** Press Keyword ***********************
```

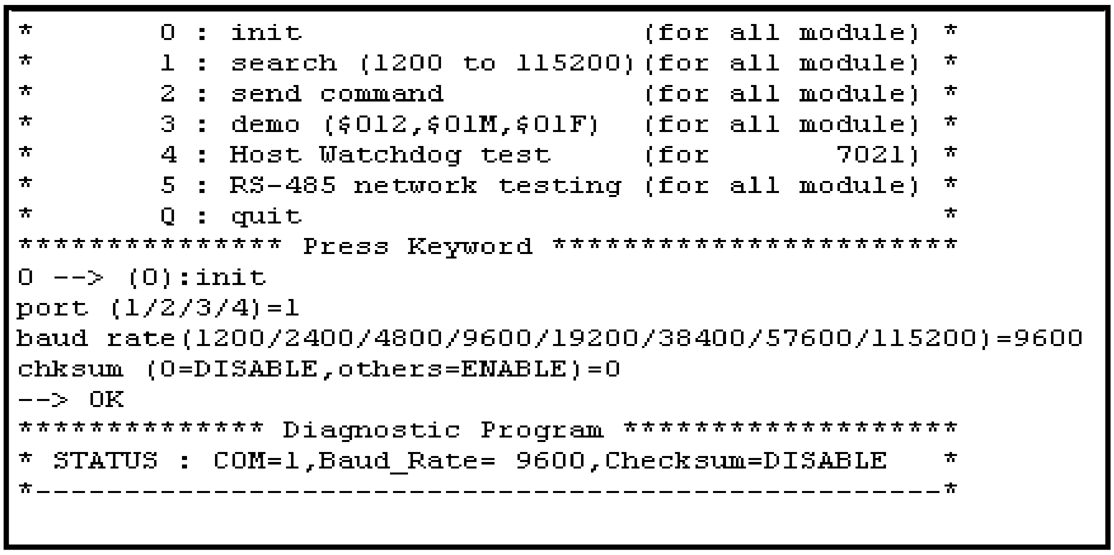
There are six functions supported in TEST.EXE given as following:

- ***0* : init.**→ to initialize the COM port
- **1 : search** → to search the 7000/8000/87000 modules in the RS-485 network
- **2 : send command** → to send command string and receive the result string
- **3 : demo** → to demo command "$012","$01M","$01F"
- **4 : Host Watchdog Test** → test programmable host watchdog
- **5 : RS-485 network testing** → test the stability of RS-485 network
- **Q: quit** → to quit the TEST.EXE

2.1.1 Init Function

The operation steps to_initialize the COM port are given as following :

1. step 1 : press **0**
2. step 2 : press **1[Enter]**
3. step 3 : press **9600[Enter]**
4. step 4 : press **0[Enter]**



```
*       0 : init                    (for all module) *
*       1 : search (1200 to 115200)(for all module) *
*       2 : send command            (for all module) *
*       3 : demo ($012,$01M,$01F)   (for all module) *
*       4 : Host Watchdog test      (for       7021) *
*       5 : RS-485 network testing (for all module) *
*       Q : quit                                     *
*************** Press Keyword ***********************
0 --> (0):init
port (1/2/3/4)=1
baud rate(1200/2400/4800/9600/19200/38400/57600/115200)=9600
chksum (0=DISABLE,others=ENABLE)=0
--> OK
*************** Diagnostic Program *******************
* STATUS : COM=1,Baud_Rate= 9600,Checksum=DISABLE    *
*----------------------------------------------------*
```

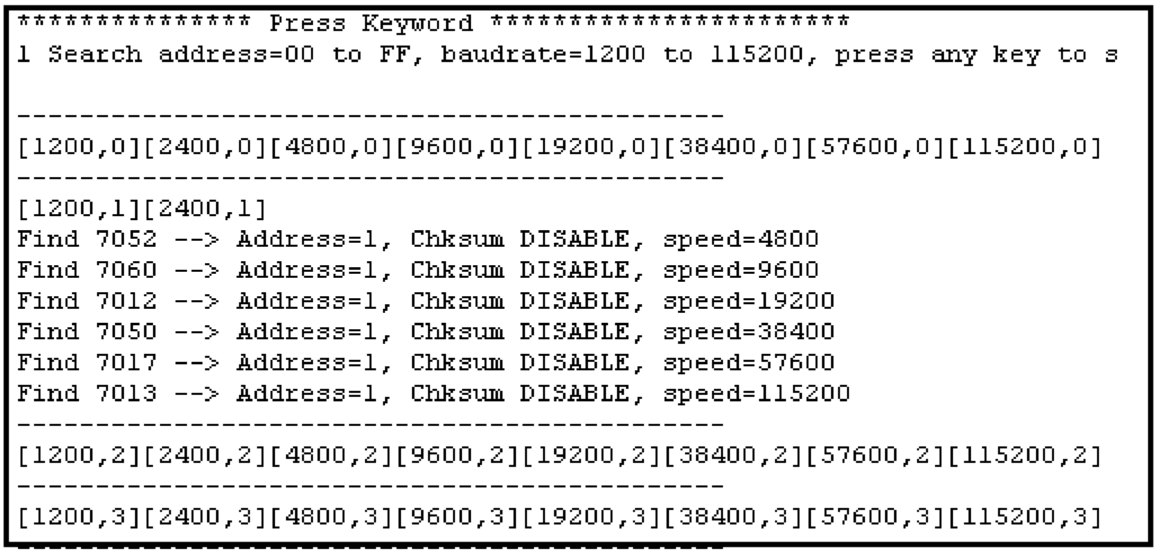
- step 1 : select functoion_0 of TEST.EXE
- step 2 : key in COM port code, 1→ COM1, 2 → COM 2, 3→COM3, 4→ COM4
- step 3 : key in baud rate, 1200 or 2400 or 4800 or 9600 or 19200 or 38400 or 57600 or 115200
- step 4 : key in checksum ENABLE/DISABLE, (0 is DISABLE, others=ENABLE)

This function will initialize the COM-port of PC. The function_2/3/4 will also use this setting later. The default setting is COM1, 9600 and checksum DISABLE. This configuration is suitable for 7000/8000/87000 default setting. If the 7000/8000/87000 modules are used in another configuration, the user has to use this function to re-initialize the PC COM port.

2.1.2 Search Function

The operation steps of ***search*** function all the 7000/8000/87000 modules connected in the RS-485 network are given as following:

- step 1 : press 1
- step 2 : wait until no need to search, **press any key**



```
*************** Press Keyword ***********************
1 Search address=00 to FF, baudrate=1200 to 115200, press any key to s

--------------------------------------------
[1200,0][2400,0][4800,0][9600,0][19200,0][38400,0][57600,0][115200,0]
--------------------------------------------
[1200,1][2400,1]
Find 7052 --> Address=1, Chksum DISABLE, speed=4800
Find 7060 --> Address=1, Chksum DISABLE, speed=9600
Find 7012 --> Address=1, Chksum DISABLE, speed=19200
Find 7050 --> Address=1, Chksum DISABLE, speed=38400
Find 7017 --> Address=1, Chksum DISABLE, speed=57600
Find 7013 --> Address=1, Chksum DISABLE, speed=115200
--------------------------------------------
[1200,2][2400,2][4800,2][9600,2][19200,2][38400,2][57600,2][115200,2]
--------------------------------------------
[1200,3][2400,3][4800,3][9600,3][19200,3][38400,3][57600,3][115200,3]
--------------------------------------------
```

This function will search from baud rate 1200 to 115200 for address 00. Then search from baud rate 1200 to 115200 for address 01, …… At last search form baud rate 1200 to 115200 for address 0xFF. If the user press any key during the search interval, this function will stop immediately. If any 7000/8000/87000 module is found, the **module name, address, checksum status and baud rate** will be shown in the screen. In this example, the search function find six 7000/8000/87000 module in the RS-485 network. Their module addresses are all equal to 01 but their baud rate are all different.

The 7000/8000/87000 series modules can **share the same RS-485 network with different baud rate.** Therefor there can be installed 256*8=2048 modules with repeater in one RS-485 network. This function will search all possible modules including checksum enable/disable in the RS-485 network, therefore this function will search **2048*2=4096** times max. in one completely search interval as following:

1. step 1 : baud rate=1200, send **$002[Enter]** → wait for response
2. step 2 : baud rate=1200, send **$002B6[Enter]**→ wait for response
3. step 3 : baud rate=2400, send **$002[Enter]** → wait for response
4. step 4 : baud rate=2400, send **$002B6[Enter]**→ wait for response
5. ……………………………………………………………………………..
6. step 15 : baud rate=115200, send **$002[Enter]** → wait for response
7. step 16 : baud rate=115200, send **$002B6[Enter]**→ wait for response
8. step 17 : baud rate=1200, send **$012[Enter]** → wait for response
9. step 18 : baud rate=1200, send **$012B7[Enter]** → wait for response
10. ……………………………………………………………………………..
11. ……………………………………………………………………………..
12. step 31 : baud rate=115200, send **$012[Enter]** → wait for response
13. step 32 : baud rate=115200, send **$012B7[Enter]**→ wait for response
14. ……………………………………………………………………………..
15. ……………………………………………………………………………..
16. **step 4095** : baud rate=115200, send **$FF2[Enter]**
17. **step 4096** : baud rate=115200, send **$FF2E2[Enter]**

for address 00

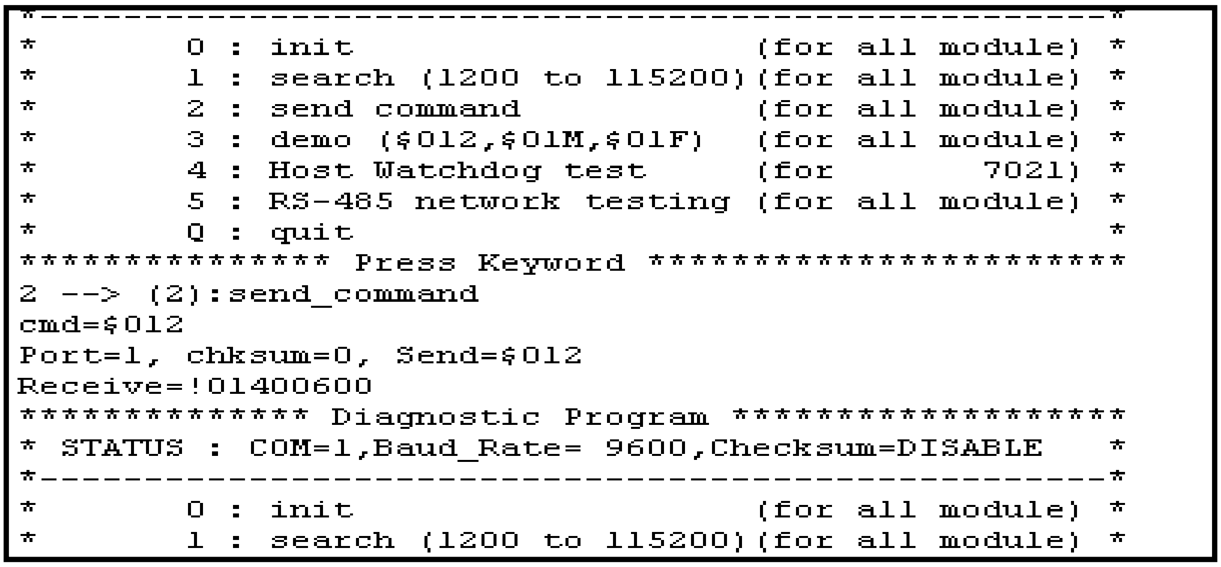
for address 01

for address FF

2.1.3 Send Command Function

The operation steps of **send command** function are given as following:

1. step 1 : press **2**
2. step 2 : press **??…??[Enter]**



```
*-----------------------------------------------------------*
*        0 : init                    (for all module) *
*        1 : search (1200 to 115200)(for all module) *
*        2 : send command            (for all module) *
*        3 : demo ($012,$01M,$01F)   (for all module) *
*        4 : Host Watchdog test      (for       7021) *
*        5 : RS-485 network testing (for all module) *
*        Q : quit                                     *
*************** Press Keyword ***********************
2 --> (2):send_command
cmd=$012
Port=1, chksum=0, Send=$012
Receive=!01400600
************** Diagnostic Program *******************
* STATUS : COM=1,Baud_Rate= 9600,Checksum=DISABLE    *
*-----------------------------------------------------------*
*        0 : init                    (for all module) *
*        1 : search (1200 to 115200)(for all module) *
```

The operation steps of this function are given as following:

Step 1: This function will automatically add the **[0x0D]** to the input string.

Step 2 : If the checksum status is ENABLE, the extra two checksum bytes will be added into the input string also.

Step 3: This function will send out the command string first then wait for the result string.

Step 4: If the result string is received before the time-out interval, this function will automatically check the format of the result string. The result string will be shown on the screen.

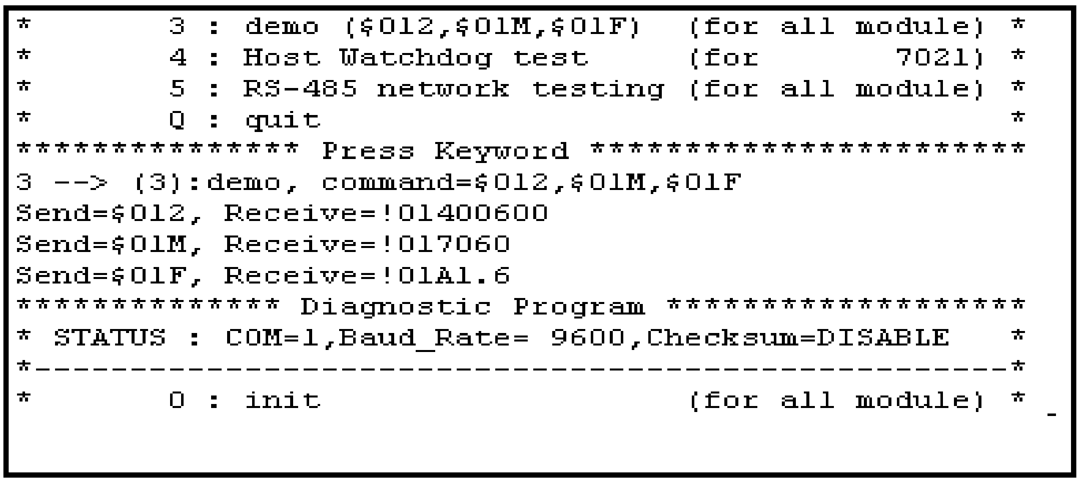
Step 5: If no module response to this command, the time-out message will be shown on the screen.

2.1.4 Demo Function

$012[0x0D]	: read module configuration code
$01M[0x0D]	: read module name
$01F[0x0D]	: read firmware version number

This function will send out these three commands in sequence to identify the module's basic information. This function is very useful during the diagnostic interval.

The operation step of ***demo*** function is given as following:
    step 1 : press **3**

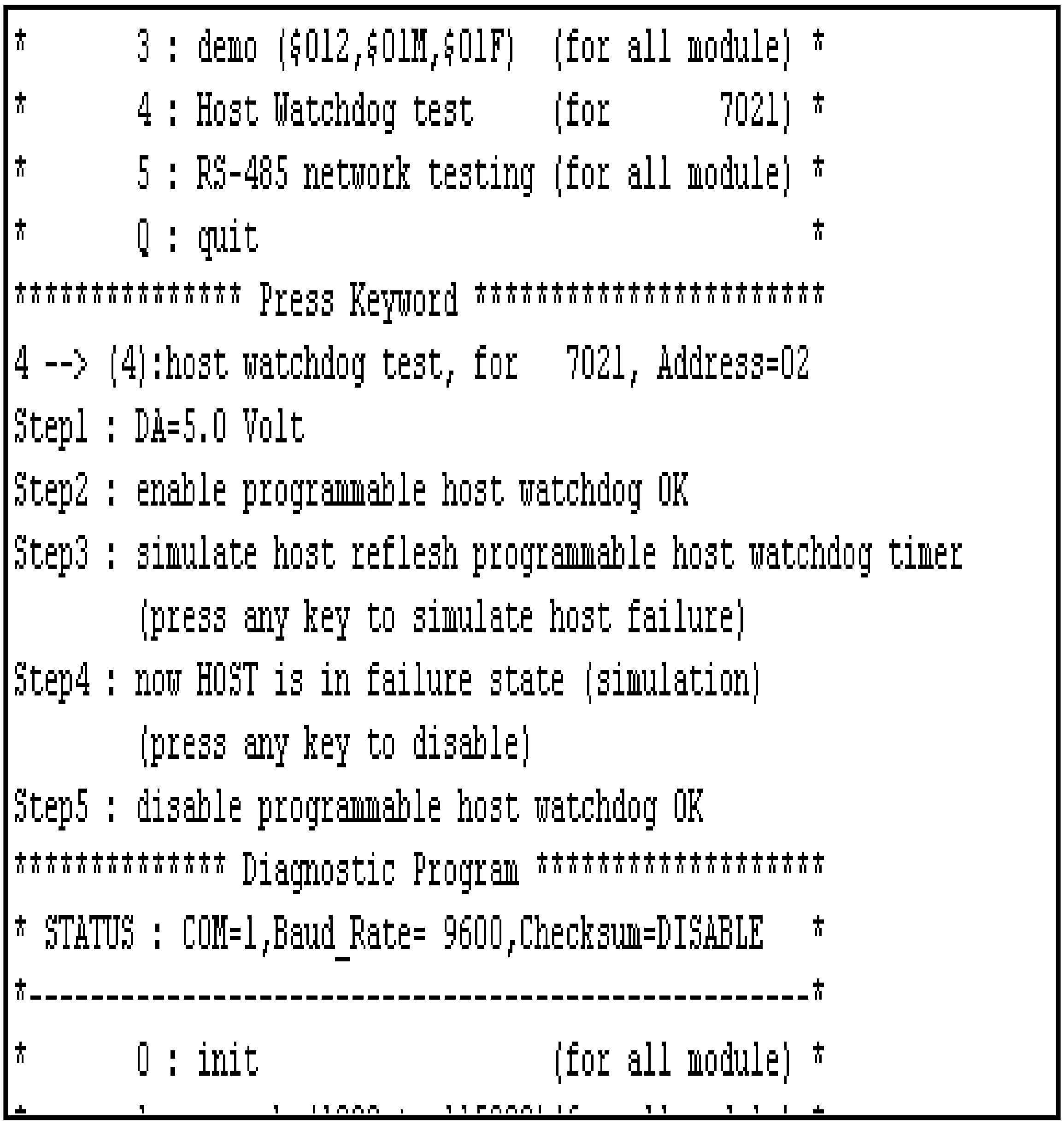


```
*        3 : demo ($012,$01M,$01F)   (for all module) *
*        4 : Host Watchdog test      (for        7021) *
*        5 : RS-485 network testing (for all module) *
*        Q : quit                                      *
*************** Press Keyword ***********************
3 --> (3):demo, command=$012,$01M,$01F
Send=$012, Receive=!01400600
Send=$01M, Receive=!017060
Send=$01F, Receive=!01A1.6
************** Diagnostic Program *******************
* STATUS : COM=1,Baud_Rate= 9600,Checksum=DISABLE    *
*---------------------------------------------------*
*        0 : init                    (for all module) * _
```

2.1.5 Test Host WatchDog Function

The operation steps to **test programmable host watchdog** are shown as following:

1. step 1 : press **4**
2. step 2 : **press any key** to simulate host failure
3. step 3 : **press any key** to clear module status



```
*       3 : demo ($012,$01M,$01F)  (for all module) *
*       4 : Host Watchdog test     (for       7021) *
*       5 : RS-485 network testing (for all module) *
*       Q : quit                                    *
*************** Press Keyword **********************
4 --> (4):host watchdog test, for   7021, Address=02
Step1 : DA=5.0 Volt
Step2 : enable programmable host watchdog OK
Step3 : simulate host reflesh programmable host watchdog timer
        (press any key to simulate host failure)
Step4 : now HOST is in failure state (simulation)
        (press any key to disable)
Step5 : disable programmable host watchdog OK
************** Diagnostic Program ******************
* STATUS : COM=1,Baud_Rate= 9600,Checksum=DISABLE   *
*---------------------------------------------------*
*       0 : init                   (for all module) *
```

This function gives a short demo about the programmable host watchdog. The operation steps are given as following:

step 1: send command to control D/A of 7021 to
5V(address=03).

step 2: enable the host watchdog(host watchdog timer =1
second)

step 3: send ~**[0x0D] command to refresh the host
watchdog timer

step 4: user press any key to stop the refresh operation →
simulate host failure

step 5: after 1 second, the host watchdog timer will be active
→ the DA output will go to the safe value
→ the module-LED will flash
→ the module status will be setting to 04
→ disable the host watchdog timer automatically.
→ will ignore any D/A output command from host

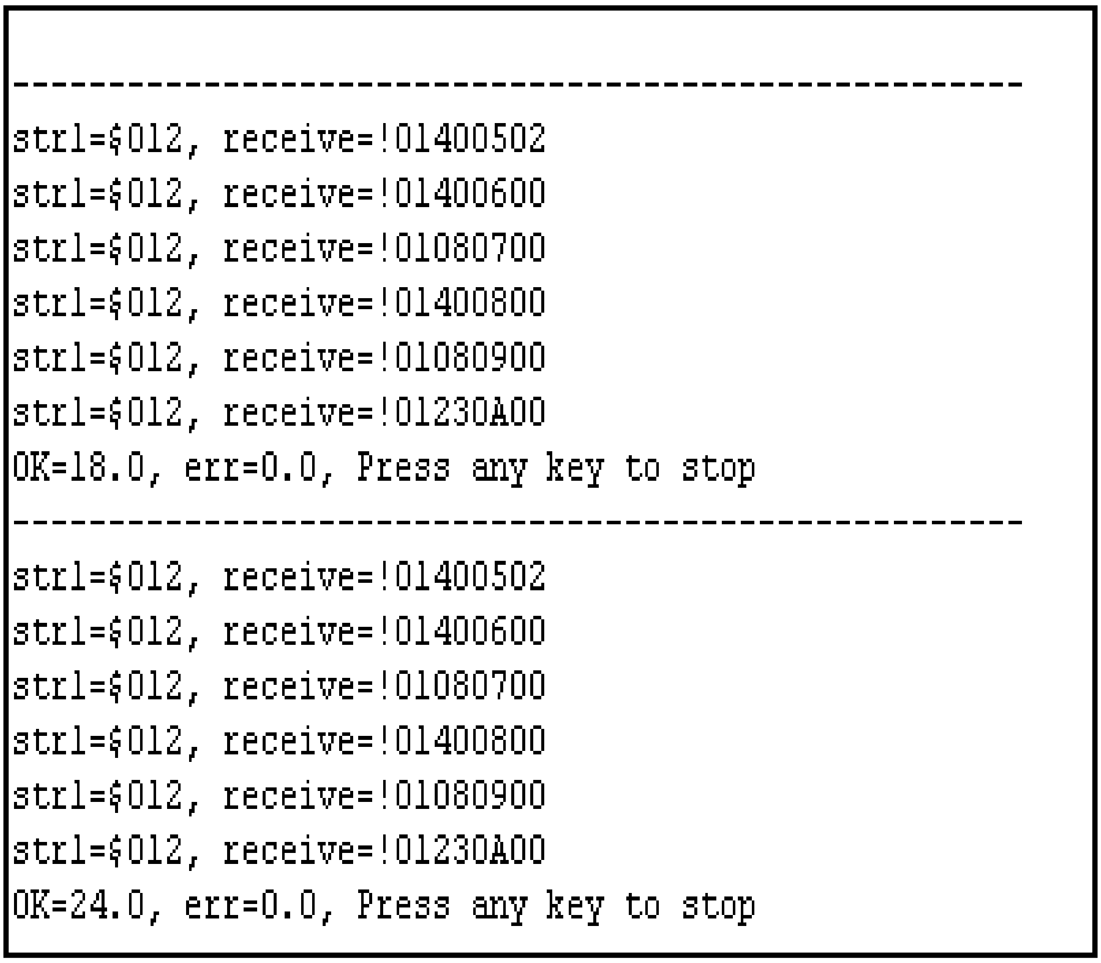
step 6 : user press any key to clear the module status
→ the module status is clear to 0
→ the module-led will no more flash
→ the DA output unchanged
→ will accept and execute host D/A output command

2.1.6 Test RS-485 Network Function

The operation steps to ***test RS-485 network*** are shown as following:

1. step 1 : install all modules into RS-485 network
2. step 2 : edit **TEST.DAT**
3 step 3 : **press 5** to select "RS-485 network testing" function

This function will read the command and expected response from TEST.DAT first, then send out the command and check the module response. If the module response and expected response is the same → OK=OK+1. If they are different → ERR=ERR+1. This function will repeat testing until the user press any key to stop



```
--------------------------------------------------------
str1=$012, receive=!01400502
str1=$012, receive=!01400600
str1=$012, receive=!01080700
str1=$012, receive=!01400800
str1=$012, receive=!01080900
str1=$012, receive=!01230A00
OK=18.0, err=0.0, Press any key to stop
--------------------------------------------------------
str1=$012, receive=!01400502
str1=$012, receive=!01400600
str1=$012, receive=!01080700
str1=$012, receive=!01400800
str1=$012, receive=!01080900
str1=$012, receive=!01230A00
OK=24.0, err=0.0, Press any key to stop
```

The format of TEST.DAT is given as following:

```
line 1 : N → number of commands to test
line 2 : command_1
line 3 : command_2
…………………..
line N+1 : command_N
```

The format of command is given as following:

com port	Baud rate	checksum	Comman d	expected response
1/2/3/4	1200/2400/4800/9600/ 19200/38400/57600/ 115200	0 : disable 1 : enable	7000 command	expected response from 7xxx

The default TEST.DAT is given as following :

```
6
1   4800  0  $012  !01400502
1   9600  0  $012  !01400600
1  19200  0  $012  !01080700
1  38400  0  $012  !01400800
1  57600  0  $012  !01080900
```

2.2 DIAG TC Source Program

The usage of TEST.EXE are given in Sec. 2.1. The complete source codes are given in the companion floppy disk. These programs are written in TURBO-C.

The files in \DCON_Utility_DOS\DIAG\TC are given as below:

1. UART.H	→ header file
2. UART.C	→ library file
3. TEST.C	→ application file
4. TEST.PRJ	→ project file

The compiler and link for TC are given as below:

1. Set DOS PATH to TC
2. Using TC integrated environment to set ***PROJECT*** to TEST.PRJ
3. Press ***F9*** to MAKE the project file (TEST.PRJ)
4. Execution file = ***TEST.EXE***

2.2.1 TEST.PRJ

```
UART.C
TEST.C
```

2.2.2 TEST.DAT

```
6
1   4800  0  $012  !01400502
1   9600  0  $012  !01400600
1  19200  0  $012  !01080700
1  38400  0  $012  !01400800
1  57600  0  $012  !01080900
1 115200  0  $012  !01230A00
```

2.2.3 UART.H

```
#include   <stdio.h>
#include   <stdlib.h>
#include   <math.h>
#include   <conio.h>
#include   <io.h>
#include   <dos.h>
#include   <stdarg.h>
#include   <string.h>

#define Com1   0x3f0
#define Com2   0x2f0
#define Com3   0x3e0
#define Com4   0x2e0

#define Txbuf   0x08     /* tx buffer */
#define Rxbuf   0x08     /* rx buffer */
#define Dll     0x08     /* baud lsb */
#define Dlh     0x09     /* baud msb */
#define Ier   0x09       /* int enable reg */
#define Fcr     0x0a     /* FIFO control register */
#define Lcr     0x0b     /* line control reg */
#define Dfr     0x0b     /* Data format  reg */
#define Mcr     0x0c     /* modem control reg */
#define Lsr     0x0d     /* line status reg */


int OPEN_COM(int iPort, long int lBaudRate);
int CLOSE_COM(int iPort);
int SEND_CMD(int iPort, char cCmd[], long int lTimeout, int
                  iChksum);
int RECEIVE_CMD(int iPort, char cCmd[], long int lTimeout,
                      int iChksum);
```

2.2.4 UART.C

```
#include   "uart.h"

int  iBase;       /* com port base address */
char cHI,cLO;
void compute_chksum(char cBuf[]);
char hex_to_ascii(int iHex);

/* ---------------------------------------------------------------------- */
/* input: port = 1/2/3/4
        baud  = 1200/2400/4800/9600/19.2K/38.4K/57.6K/115.2K
      chksum = 0 is DISABLE, others is ENABLE

  return :  0    → OK
            1    → port value error
            2    → baud rate  error
*/

int OPEN_COM(int iPort, long int lBaudRate)
{
int uart,i;
char ratehi,ratelo;
long int bb;
int cc;

switch(iPort)
    {
    case 1 : iBase=Com1; break;
    case 2 : iBase=Com2; break;
    case 3 : iBase=Com3; break;
    case 4 : iBase=Com4; break;
    default: printf("(iPort=%d)",iPort); getch();
         return 1;        /* port must 1/2/3/4 */
    }
switch(lBaudRate)
    {
    case 1200L : break;
    case 2400L : break;
    case 4800L : break;
    case 9600L : break;
    case 19200L: break;
    case 38400L: break;
    case 57600L: break;
    case 115200L: break;
    default    : return 2;          /* baud rate error */
    }
bb=115200L;
cc=bb/lBaudRate;
ratehi=(cc&0xff00)>>8;
ratelo=cc&0xff;
outportb(iBase+Lcr,0x80);     /* set DLAB */
outportb(iBase+Dll,ratelo);
outportb(iBase+Dlh,ratehi);
outportb(iBase+Lcr,0x03);     /* data format */
                    /* 0000 0011 --> 8_bit+1_stop+no_parity */
outportb(iBase+Ier,0);           /* disable interrupt */
outportb(iBase+Fcr,0x07);     /* clear input/output FIFO*/

for (i=0; i<8; i++) inportb(iBase+i);   /* clear R */
outportb(iBase+Fcr,0x01);     /* enable FIFO */
return 0;
}
/* --------------------------------------------------------------------- */
/* return : 0 --> OK
         1 --> error
*/
int CLOSE_COM(int iPort)
{
/* use polling, so do nothing here */
return 0;
}
```

```
/* -------------------------------------------------------------------- */
/* return : 0 --> OK
        1 --> port value error
        2 --> timeout
*/
int SEND_CMD(int iPort, char cCmd[], long int lTimeout, int
iChksum)
{
int i,ret;
long t;
switch(iPort)
     {
     case 1 : iBase=Com1; break;
     case 2 : iBase=Com2; break;
     case 3 : iBase=Com3; break;
     case 4 : iBase=Com4; break;
     default: return 1;        /* port must 1/2/3/4 */
     }
compute_chksum(cCmd);/* add chksum and 0x0d to input */
i=0; t=0;
while (cCmd[i]!=NULL)
{
while((inportb(iBase+Lsr)&0x20)==0)  /* check line ready */
     {
     t++; if (t>lTimeout) return 2; /* time out */
     }
outportb(iBase+Txbuf,cCmd[i]);
i++;                        /* next char */
t=0;                        /* reset timer */
}
if (iChksum!=0)
   {
   while((inportb(iBase+Lsr)&0x20)==0)      /* check line ready */
     {
     t++; if (t>lTimeout) return 2; /* time out */
     }
   outportb(iBase+Txbuf,cHI);     /* send out checksum HIGH byte */
t=0;                    /* reset timer */
while((inportb(iBase+Lsr)&0x20)==0)  /* check line ready */
     {
     t++; if (t>lTimeout) return 2; /* time out */
     }
   outportb(iBase+Txbuf,cLO);    /* send out checksum LOW byte */
   t=0;                 /* reset timer */
   }
while((inportb(iBase+Lsr)&0x20)==0)  /* check line ready */
   {
   t++; if (t>lTimeout) return 2;    /* time out */
   }
outportb(iBase+Txbuf,0x0D);         /* send out 0x0D */
return 0;                    /* send cmd OK */
}
/* ------------------------------------------------------------------- */
/* return : 0 --> OK
         1 --> port value error
         2 --> timeout
         3 --> chksum error
*/
int RECEIVE_CMD(int iPort, char cCmd[], long int lTimeout,
int iChksum)
{
int i;
char c;
long t;
switch(iPort)
     {
     case 1 : iBase=Com1; break;
     case 2 : iBase=Com2; break;
     case 3 : iBase=Com3; break;
     case 4 : iBase=Com4; break;
     default: return 1;       /* port must 1/2/3/4 */
     }
```

```
i=0; t=0;
for(;;)
{
while((inportb(iBase+Lsr)&0x01) != 0x01)   /* check line ready
*/
     {
     t++; if (t>lTimeout) return 2;    /* time_out */
     }
c=inportb(iBase+Rxbuf)&0xff;   /* receive next char */
if (c==0x0d) break;    /* receive 0x0d --> end of command */
else cCmd[i]=c;         /* store the command */
i++;                    /* next char */
if (i>35)
   {
   cCmd[i]=0;              /* string must terminate with 0 */
   printf("[%s]\7\7\7",cCmd);
   break;
   }
t=0;                    /* reset timer */
}
cCmd[i]=0;              /* string must terminate with 0 */
if (iChksum!=0) return chk_chksum(cCmd); else return 0;
}
/* ---------------------------------------------------------------------- */
void compute_chksum(char cBuf[])
{
unsigned int i,j,len,sum;
len=strlen(cBuf);
j=len;
sum=0;
for (i=0; i<len; i++) sum+=(cBuf[i]&0xff);  /* 0x0D not
included */
sum=sum&0xff;
cHI=hex_to_ascii(sum/16); /* CHKSUM high byte */
cLO=hex_to_ascii(sum%16); /* CHKSUM low  byte */
}
```

```
char hex_to_ascii(int iHex)
{
if (iHex<10) return(iHex+'0');
else return('A'+iHex-10);
}

/* ---------------------------------------------------------------------- */
/* return : 3 --> chksum error
         0 --> OK
*/

int chk_chksum(char cBuf[])
{
unsigned int i,len,sum;
char h,l;

len=strlen(cBuf); sum=0;
for (i=0; i<len-2; i++) sum+=(cBuf[i])&0xff;
sum=sum&0xff;
h=hex_to_ascii(sum/16);
l=hex_to_ascii(sum%16);

if (cBuf[len-2]!=h) return 3;  /* compare CHECKSUM high byte */
if (cBuf[len-1]!=l) return 3;   /* compare CHECKSUM low  byte */
return 0;
}
```

The UART.C includes three basic subroutines OPEN_COM, SEND_CMD and RECEIVE_CMD for 7000 modules. This file can be viewed as the library file. The detail descriptions of these three libraries are given from. Sec. 2.2.3.1 to Sec. 2.2.3.3. The application user can call these basic libraries to simply their programming design.

2.2.4.1 OPEN_COM

- **Description:**

This library is designed to initialize the COM port. The program code will directly control the PC-AT compatible UART I/O port instead of calling system BIOS utility. This code is compatible with 8250/16450/16550 UART controller. If the 16550 is used, the on-chip 16 bytes FIFO will increase the system performance.

- **Syntax:**

```
int OPEN_COM(int iPort, long lBaudRate)
```

- **Input Parameter:**

iPort: 1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate
lBaudRate: the validate baud rate are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

- **Return Value:**

0: OK
1: port value error (validate value= 1/2/4/8, others= invalidate)
2: baud rate error (validate value = 1200 / 2400 / 4800 / 9600 / 19200 / 38400 / 57600/ 115200, others = invalidate)

- **Demo Program :**
Refer to Sec. 2.2.4

2.2.4.2 SEND_CMD

- **Description :**

This library is designed to send the command string to the RS-485 network. The function will automatically add the two checksum bytes to the original string if the checksum status is ENABLE. Also this function will add the [0x0D], command terminator, to the end of the command string.

- **Syntax :**

```
int  SEND_CMD(int iPort, char cCmd[], long int lTimeout,
        int iChecksum)
```

- **Input Parameter :**

iPort: 1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate
cCmd: the starting address of the original command string (terminated with 0)
lTimeout: constant for time-out control
iChecksum: 0=DISABLE, others=ENABLE

- **Return Value:**

1 : port value error (validate value = 1/2/4/8, others = invalidate)
2 : time-out

- **Demo Program:**

Refer to Sec. 2.2.4

2.2.4.3 RECEIVE_CMD

- **Description:**

  This library is designed to receive the result string from the RS-485 network. The function will automatically check the two checksum bytes of the result string when the checksum status is ENABLE. Also this function will check the [0x0D], command terminator, in the end of the result string.

- **Syntax:**

  int RECEIVE_CMD(int iPort, char cCmd[], long int lTimeout, int iChecksum)

- **Input Parameter:**

  iPort: 1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate
  cCmd: the starting address of the result string (terminated with 0)
  lTimeout: constant for time-out control
  iChecksum: 0=DISABLE, others=ENABLE

- **Return Value:**

  0: OK
  1: port value error (validate value = 1/2/4/8, others = invalidate)
  2: time-out

- **Demo Program:**

  Refer to Sec. 2.2.4

2.2.5 TEST.C

```
#include   "uart.h"

FILE *stream;

#define   TIMEOUT  60000L

int iComPort,iChksum;
long int lBaudRate;
float ok_count[256],err_count[256];

/* ---- main -------------------------------------------------------------- */

main()
{
char cChar;

iComPort=1; iChksum=0; lBaudRate=9600L;
OPEN_COM(iComPort,lBaudRate);    /* default */
for(;;)
{
printf("\n************** Diagnostic Program *******************");
show_status();
printf("\n*      0 : init                   (for all module) *");
printf("\n*      1 : search (1200 to 115200)(for all module) *");
printf("\n*      2 : send command           (for all module) *");
printf("\n*      3 : demo ($012,$01M,$01F)  (for all module) *");
printf("\n*      4 : Host Watchdog test     (for       7021) *");
printf("\n*      5 : RS-485 network testing (for all module) *");
printf("\n*      Q : quit                                    *");
printf("\n*************** Press Keyword ***********************");
printf("\n");
cChar=getche();
switch (cChar)
      {
      case '0': init(); break;
      case '1': search(); break;
      case '2': send_command(); break;
      case '3': demo(); break;
```

```
        case '4': host_watchdog(); break;
        case '5': test_485(); break;
        case 'q':
        case 'Q': goto ret_label;
        default : printf(" --> Error Keyword"); break;
        }

}
ret_label:
printf("\n************** Diagnostic Program *******************");
}

/* ---- show status ----------------------------------------------------- */

show_status()
{
printf("\n* STATUS : COM=%d,",iComPort);
printf("Baud_Rate=%5ld,",lBaudRate);
if (iChksum==0) printf("Checksum=DISABLE   *");
else printf("Checksum=ENABLE    *");
printf("\n*--------------------------------------------------*");
}

/* ---- function 0 ------------------------------------------------------ */

init()
{
int iRet,iPort;

printf(" --> (0):init\n");
printf("port (1/2/3/4)="); scanf("%d",&iPort);
printf("baud rate(1200/2400/4800/9600/19200/38400/57600/115200)=");
scanf("%ld",&lBaudRate);
printf("chksum (0=DISABLE,others=ENABLE)="); scanf("%d",&iChksum);
iRet=OPEN_COM(iPort,lBaudRate);
if (iRet==0) {printf("--> OK"); iComPort=iPort;}
else if (iRet==1) printf("--> port error");
else if (iRet==2) printf("--> baudrate error");
}
```

```
/* ---- function 1 ------------------------------------------------------ */
char cCmd1[100],cCmd2[100],cCmd3[100];
search()
{
int i,iD1,iD2,iRet,j;
long BaudRate;
char cFlag;

printf(" Search address=00 to FF, baudrate=1200 to 115200, press any key to stop\n");

cCmd1[0]='$';
strcpy(cCmd3,"Test Command");

for(;;)
for (i=0; i<256; i++)        /* search from adress_00 to adress_FF */
   {
   printf("\n---------------------------------------------\n");
   for (j=3; j<=10; j++) /* search from BaudRate_1200 to BaudRate_115200 */
      {
      switch (j)
           {
           case 3 : BaudRate=1200L; break;
           case 4 : BaudRate=2400L; break;
           case 5 : BaudRate=4800L; break;
           case 6 : BaudRate=9600L; break;
           case 7 : BaudRate=19200L; break;
           case 8 : BaudRate=38400L; break;
           case 9 : BaudRate=57600L; break;
           case 10: BaudRate=115200L; break;
           }

/*    OPEN_COM(iComPort,BaudRate); */

iRet=OPEN_COM(iComPort,BaudRate);
if (iRet!=0) printf(" Open Error, iComPort=%d, BaudRate=%ld
iRet=%d",iComPort,BaudRate,iRet);
SEND_CMD(iComPort,cCmd3,TIMEOUT,0); /* RS-232 settling time delay */
RECEIVE_CMD(iComPort,cCmd2,TIMEOUT,0); /* chksum disable */
```

```
     iD1=i/16; iD2=i%16;
     cCmd1[1]=get_ascii(iD1);
     cCmd1[2]=get_ascii(iD2);
     cCmd1[3]='2'; cCmd1[4]=0;  /* $AA2 --> read status */

     cFlag=0;

/* ------------------- checksum DISABLE ---------------------------------- */

     SEND_CMD(iComPort,cCmd1,TIMEOUT,0);          /* chksum disable */
     iRet=RECEIVE_CMD(iComPort, cCmd2, TIMEOUT,0);  /* chksum disable */
     if (iRet==0)
       {
       cFlag=1;                  /* find module in chksum DISABLE */
       cCmd1[3]='M'; cCmd1[4]=0; /* $AAM --> read module name */
       SEND_CMD(iComPort,cCmd1,TIMEOUT,0);        /* chksum disable */
       iRet=RECEIVE_CMD(iComPort, cCmd2, TIMEOUT,0);/* chksum disable */
       if (iRet==0)
         {
         printf("\nFind %c%c%c%c --> Address=%d, Chksum DISABLE, speed=%ld ",
               cCmd2[3],cCmd2[4],cCmd2[5],cCmd2[6],i,BaudRate);
         }
       }

/* ------------------- checksum ENABLE----------------------------------- */

     cCmd1[3]='2'; cCmd1[4]=0;   /* $AA2 --> read status */
     SEND_CMD(iComPort,cCmd1,TIMEOUT,1);          /* chksum  enable */
     iRet=RECEIVE_CMD(iComPort, cCmd2, TIMEOUT,1);  /* chksum  enable */
     if (iRet==0)
       {
       cFlag=1;                  /* find module in chksum  ENABLE */
       cCmd1[3]='M'; cCmd1[4]=0; /* $AAM --> read module name */
       SEND_CMD(iComPort,cCmd1,TIMEOUT,1);        /* chksum  enable */
       iRet=RECEIVE_CMD(iComPort, cCmd2, TIMEOUT,1);/* chksum  enable */
       if (iRet==0)
         {
         printf("\nFind %c%c%c%c --> Address=%d, Chksum ENSABLE, speed=%ld ",
               cCmd2[3],cCmd2[4],cCmd2[5],cCmd2[6],i,BaudRate);
         }
       }
```

```
        if (cFlag==0) printf("[%ld,%x]",BaudRate,i);
        if (kbhit()!=0) {getch(); goto ret_label;}
        }
    }

ret_label:
OPEN_COM(iComPort,lBaudRate);     /* default */
}

/* ------------------------------------------------------------------- */

get_ascii(int iHex)
{
if (iHex<10) return(iHex+'0');
else return('A'+iHex-10);
}

/* ---- function 2 ------------------------------------------------------ */

send_command()
{
char cCmd1[50],cCmd2[50];
int iRet;

printf(" --> (2):send_command\n");
printf("cmd="); scanf("%s",cCmd1);
SEND_CMD(iComPort,cCmd1,TIMEOUT,iChksum);
iRet=RECEIVE_CMD(iComPort, cCmd2, TIMEOUT,iChksum);
printf("Port=%d, chksum=%d, Send=%s\n",iComPort,iChksum,cCmd1);
if (iRet==0) printf("Receive=%s",cCmd2);
else if (iRet==1) printf("Receive=com value error (must 1/2/3/4)");
else if (iRet==2) printf("Receive=Timeout");
else if (iRet==3) printf("Receive=chksum error");
}

/* ---- function 3 ------------------------------------------------------ */

demo()
{
char cCmd[50];
int iRet;
```

```
printf(" --> (3):demo, command=$012,$01M,$01F");

strcpy(cCmd,"$012");
printf("\nSend=$012, ",iComPort);
SEND_CMD(iComPort,cCmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
if (iRet==0) printf("Receive=%s",cCmd);
else if (iRet==1) printf("Receive=com value error (must 1/2/3/4)");
else if (iRet==2) printf("Receive=Timeout");
else if (iRet==3) printf("Receive=chksum error");

strcpy(cCmd,"$01M");
printf("\nSend=$01M, ",iComPort);
SEND_CMD(iComPort,cCmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
if (iRet==0) printf("Receive=%s",cCmd);
else if (iRet==1) printf("Receive=com value error (must 1/2/3/4)");
else if (iRet==2) printf("Receive=Timeout");
else if (iRet==3) printf("Receive=chksum error");

strcpy(cCmd,"$01F");
printf("\nSend=$01F, ",iComPort);
SEND_CMD(iComPort,cCmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
if (iRet==0) printf("Receive=%s",cCmd);
else if (iRet==1) printf("Receive=com value error (must 1/2/3/4)");
else if (iRet==2) printf("Receive=Timeout");
else if (iRet==3) printf("Receive=chksum error");
}
/* ---- function 4 ------------------------------------------------------ */

host_watchdog()
{
char cCmd[50];
int iRet;

printf(" --> (4):host watchdog test, for   7021, Address=03");

strcpy(cCmd,"#0305.000");
SEND_CMD(iComPort,cCmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
```

```
if ((iRet==0) & (cCmd[0]=='>'))
  printf("\nStep1 : DA=5.0 Volt");
else {printf("\ntest error"); return;}

strcpy(cCmd,"~03310A");
SEND_CMD(iComPort,cCmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
if ((iRet==0) & (cCmd[0]=='!'))
  printf("\nStep2 : enable host watchdog OK");
else {printf("\ntest error"); return;}

printf("\nStep3 : HOST continuously reflesh watchdog timer (normal condition)");
printf("\n       (press any key to simulate host failure)");
for(;;)
     {
     strcpy(cCmd,"~**");
     SEND_CMD(iComPort,cCmd,TIMEOUT,0);
     iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
     if (kbhit()!=0) {getch(); break;}
     }

printf("\nStep4 : HOST is failure now (simulation)");
printf("\n       --> host watchdog will active");
printf("\n       --> D/A will go to safe state");
printf("\n       --> module LED will flash");
printf("\n       (press any key to continue)");
getch();

strcpy(cCmd,"~031");
SEND_CMD(iComPort,cCmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, cCmd, TIMEOUT,0);
if (iRet==0)
  {
  printf("\nStep5 : clear module status");
  printf("\n       --> module LED will not flash");
  }
else {printf("\ntest error"); return;}
}
```

```
/* ---- function 5 ------------------------------------------------------ */

test_485()
{
int i,j,k,num,ComPort,checksum,iRet;
long BaudRate;
char str1[80],str2[80],str3[80];
float ok,err;
ok=err=0.0;
for(;;)
{
stream=fopen("test.dat","r");
fscanf(stream,"%d",&num);

for (k=0; k<num; k++)
      {
      ok+=1.0;
      fscanf(stream,"%d %ld %d %s %s",&ComPort,&BaudRate,&checksum,str1,str2);

      iRet=OPEN_COM(ComPort,BaudRate);
      if (iRet!=0) err+=1.0;
      SEND_CMD(ComPort,"~**",TIMEOUT,checksum);  /* RS-232 settling time delay */
      RECEIVE_CMD(ComPort,str3,TIMEOUT,checksum);

      SEND_CMD(ComPort,str1,TIMEOUT,checksum);
      RECEIVE_CMD(ComPort,str3,TIMEOUT,checksum);
      printf("\nstr1=%s, receive=%s",str1,str3);
      if (memcmp(str2,str3,strlen(str2))!=0) err+=1.0;
      }

fclose(stream);
printf("\nOK=%.1f, err=%.1f, Press any key to stop",ok,err);
printf("\n--------------------------------------------------");
if (kbhit()!=0) {getch(); return;}
}
}
```

2.2.5.1 Function_0: INIT

This function is designed to initiate the COM port. The program steps are given as below:

step 1 : input the COM port number
step 2 : input the baud rate number
step 3 : input the checksum status
step 4 : call OPEN_COM to initiate the COM port
step 5 : show the result of operation

This function set the parameters for the COM port and the function_1 to function_4 will also operate based on these parameters. Refer to Sec. 2.1.1 for demonstration.

2.2.5.2 Funcrtion_1: Search

This function is designed to search the 7000 modules in the RS-485 network. The program steps are given as below:

step 1: calculate next address AA, if AA>FF → Stop
step 2: for each baud rate from 1200 to 115200, repeat step3 to step7
step 3: send out command string **$AA2[0x0D]** in checksum DISABLE
step 4: if any response in step3 → find a new 7000 module → send out command string **$AAM[0x0D]** to get the module name and show it in the screen
step 5: send out command string **$AA2(chk)[0x0D]** in checksum ENABLE
step 6: if any response in step5 → find a new 7000 module → send out command string **$AAM(chk)[0x0D]** to get the module name and show it in the screen
step 7: if user press any key → stop this function. Otherwise go to step 1 to search next address

The modules found by this function will be shown in the screen with their address and checksum status. Refer to Sec. 2.1.2 for demonstration.

2.2.5.3 Function_2: Send Command

This function is designed to send out the command string to the RS-485 network. This function will call SEND_CMD to send out command string and call the RECEIVE_CMD to receive the result string. The SEND_CMD will automatically add the two checksum bytes and the [0x0D], command terminator, to the original command string. The RECEIVE_CMD will check the [0x0D] and the two checksum bytes if needed. The program steps are given as below:

step 1: Input the original command string
step 2: CALL the SEND_CMD to send out command string
step 3: CALL the RECEIVE_CMD to receive the result string
step 4: Show the operation status and the result string if operation success

Refer to Sec. 2.1.3 for demonstration.

2.2.5.4 Function_3: Demo

This function is designed to read and show the module status, module name and module firmware version. These three parameters are the basic information for each 7000 module. The module address of the factory setting is 01, so this function is designed for the new module. The address is fixed to 01 and the checksum status is DISABLE. The program steps are given as below:

step 1 : Call SEND_CMD to send out the command string $012
step 2 : Call RECEIVE_CMD to show the module status
step 3 : Call SEND_CMD to send out the command string $01M
step 4 : Call RECEIVE_CMD to show the module name
step 5 : Call SEND_CMD to send out the command string $01F
step 6 : Call RECEIVE_CMD to show the module firmware version

Refer to Sec. 2.1.4 for demonstration.

2.2.5.5 Function_4: Test Host Watchdog

This function give a demonstration for the usage of host watchdog. The operation steps are given as below :

Step 1: send out the **#0305.000[0x0D]** command
Step 2: read back and show the operation result
step 3: enable the host watchdog timer and set the watchdog timer = 1 sec
step 4: send out the **~**[0x0D]** command to refresh the host watchdog timer until user press any key to simulate a host failure condition
step 5: wait for user's key to continue. User can use a voltage meter to measure the D/A output safe value
step 6: clear the module status

Refer to Sec. 2.1.5 for demonstration.

2.2.5.6 Function_5: RS-485 Network Test

This function is designed to test the reliability of RS-485 network. The operation steps are given as below :

step 1: read next command and test pattern from TEST.DAT, repeat step2 to step6.
step 2: initiate COM port
step 3: send out command and receive result string
step 4: compare the result string & the test pattern (read from TEST.DAT)
step 5: if (result_string = test_pattern) OK=OK+1 else Error=Error+1
step 6 : Go to step 1

Refer to Sec. 2.1.6 for demonstration.

2.3 DIAG QB Source Program

The **QB.BAS** in \NAP7000S\DIAG\QB is a QBASIC program, its source listing is given as following:

```
10 OPEN "COM1:9600,N,8,1,RS,CS,CD,DS" AS #1
20 CMD$="$012"
30 PRINT #1, CMD$
40 RESULT$=INPUT$(9,#1)
50 PRINT "Send=$012 --> Receive=",RESULT$
60 CLOSE:END
```

3 Learning Kits Software

There are six learning kits given as following:

Digital control learning kit 1: 7520 + 7060 → 4*D/I + 4*relay

Analog learning kit: 7520 + 7012D → 16-bit A/D + 5-digit LED display

RS-485 network learning kit: 7520 * 3 → for PC networking or PLC networking

High speed high number digital control kit: 7520 + 7053 + 7043 → 16*D/I + 16*D/O(O.C.)

High speed high number data acquisition kit: 7520 + 7017 + 7018 → 8*A/I + 8*thermocouple

Professional kit: 7520 + 7060 + 7012D + 7021
→ 4*D/I+4*Relay+16-bit A/D(with LED)+12-bit D/A(with V/I output)

The module status of learning-kits are given as following:

module	address	type	channel	baud rate	Checksum	Description
7060	01	40	4+4	9600	disable	4*D/I + 4*Relay
7012D	02	08	1	9600	disable	A/D, +/-10V range
7021	03	32	1	9600	disable	D/A, 0-10V voltage
7053	04	40	16	9600	disable	16*D/I(4-30V)
7063	05	40	16	9600	disable	16*D/O(O.C.)
7017	06	08	8	9600	disable	A/D, +/-10V range
7018	07	0E	8	9600	disable	A/D, J-thermocouple

3.1 Driver Source

\DCON_Utility_DOS\learning\TEST.C	→ main program file
\DCON_Utility_DOS\learning\UART.C	→ USART library file
\DCON_Utility_DOS\UART.H	→ header file
\DCON_Utility_DOS\learning\TEST.PRJ	→ TC project file
\DCON_Utility_DOS\learning\TEST.EXE	→ learning kits execution file
\DCON_Utility_DOS\learning\learning.dat	→ data file for TEST.EXE

Type “TEST.EXE” to execute learning kits utility software.

3.2 Function Descriptions

3.2.1 Function_0: INIT

Refer to Sec. 2.1.1

3.2.2 Function_1: Search

Refer to Sec. 2.1.2

3.2.3 Function_2: Send Command

Refer to Sec. 2.1.3

3.2.4 Function_3: Find Learning Kits

Send out $01M, $02M, $03M, $04M, $05M, $06M and $07M commands to show all modules of learning kits.

3.2.5 Function_4: 7060 Test

The default setting of 7060 are given as following:

- address=01, baud rate=9600, checksum=disable
- 4 channels of D/I are connected to 4 channels of D/O

The operation steps of demo program:

- step 1: turn relay-1 ON, read all D/I
- step 2: turn rely-2 ON, read all D/I
- step 3: turn relay-3 ON, read all D/I
- step 4: turn relay-4 ON, read all D/I
- repeat step1 to step4 until user press any key

3.2.6 Function_5: 7012D Test

The default setting of 7012D are given as following:

- address=02, baud rate=9600, checksum=disable, type=08=+/-10V range

The operation steps of demo program:

- step 1: read analog input
- step 2: show analog input value in screen
- repeat step1 to step2 until user press any key

3.2.7 Function_6: 7017 Test

The default setting of 7017 are given as following:

- address=06, baud rate=9600, checksum=disable, type=08=+/-10V range

The operation steps of demo program:

- step 1: read 8 channels of analog input
- step 2: show 8 channels of analog input values in screen
- repeat step1 to step2 until user press any key

3.2.8 Function_7: 7018 Test

The default setting of 7018 are given as following:

- address=07, baud rate=9600, checksum=disable, type=0E=J-type thermocouple

The operation steps of demo program:

- step 1: read 8 channels of thermocouple input
- step 2: show 8 channels of temperature values in screen
- repeat step1 to step2 until user press any key

3.2.9 Function_8: 7053/7043 Test

The default setting of 7053 are given as following:

- address=04, baud rate=9600, checksum=disable, 16 channels of D/I

The default setting of 7043 are given as following:

- address=05, baud rate=9600, checksum=disable, 16 channels of D/O
- 16 channels of D/I are connected to 16 channels of D/O

The operation steps of demo program:

- step 1: turn D/O channel_1 ON, read all D/I
- ..
- step16: turn D/O channel_16 ON, read all D/I
- repeat step1 to step16 until user press any key

3.2.10 Function_9: 7012D/7021 Test

The default setting of 7012D are given as following:

- address=02, baud rate=9600, checksum=disable, type=08=+/- 10V range

The default setting of 7021 are given as following:

- address=03, baud rate=9600, checksum=disable, type=32=0-10V voltage
- voltage output is connected to analog input of 7012D

The operation steps of demo program:

- step 1: send out D/A=1.0V, read A/D value
- ..
- step 9: send out D/A=9.0V, read A/D value
- repeat step1 to step9 until user press any key

3.2.11 Function_H: Help Information

Show help information of 7000 learning kits.

3.3 Demo of the Digital Control learning Kit

- run TEST.EXE
- press 3 → to find all modules name
- press H → to show help information
- press 4 → to test 7060
- press any key to stop function_4

3.4 Demo of the Analog learning Kit

- run TEST.EXE
- press 3 → to find all modules name
- press H → to show help information
- apply a DC voltage source to analog input of 7012D
- press 5 → to test 7012D
- press any key to stop function_5

3.5 Demo of the RS-485 Network learning Kit

- Refer to Chapter_4 for PC networking
- Refer to Chapter_5 for PLC networking

3.6 Demo of the high speed high number Digital Control Kit

- run TEST.EXE
- press 3 → to find all modules name
- press H → to show help information
- press 8 → to test 7053 & 7043
- press any key to stop function_8

3.7 Demo of the high speed high number data acquisition Kit

- run TEST.EXE
- press 3 → to find all modules name
- press H → to show help information
- press 6 → to test 7017
- press any key to stop function_6
- press 7 → to test 7018
- press any key to stop function_7

3.8 Demo of the the professional kit

- run TEST.EXE
- press 3 → to find all modules name
- press H → to show help information
- press 4 -> to test 7060
- press any key to stop function_4
- press 9 → to test 7012D & 7021
- press any key to stop function_9

4 PC Networking Applications

4.1 Introduction

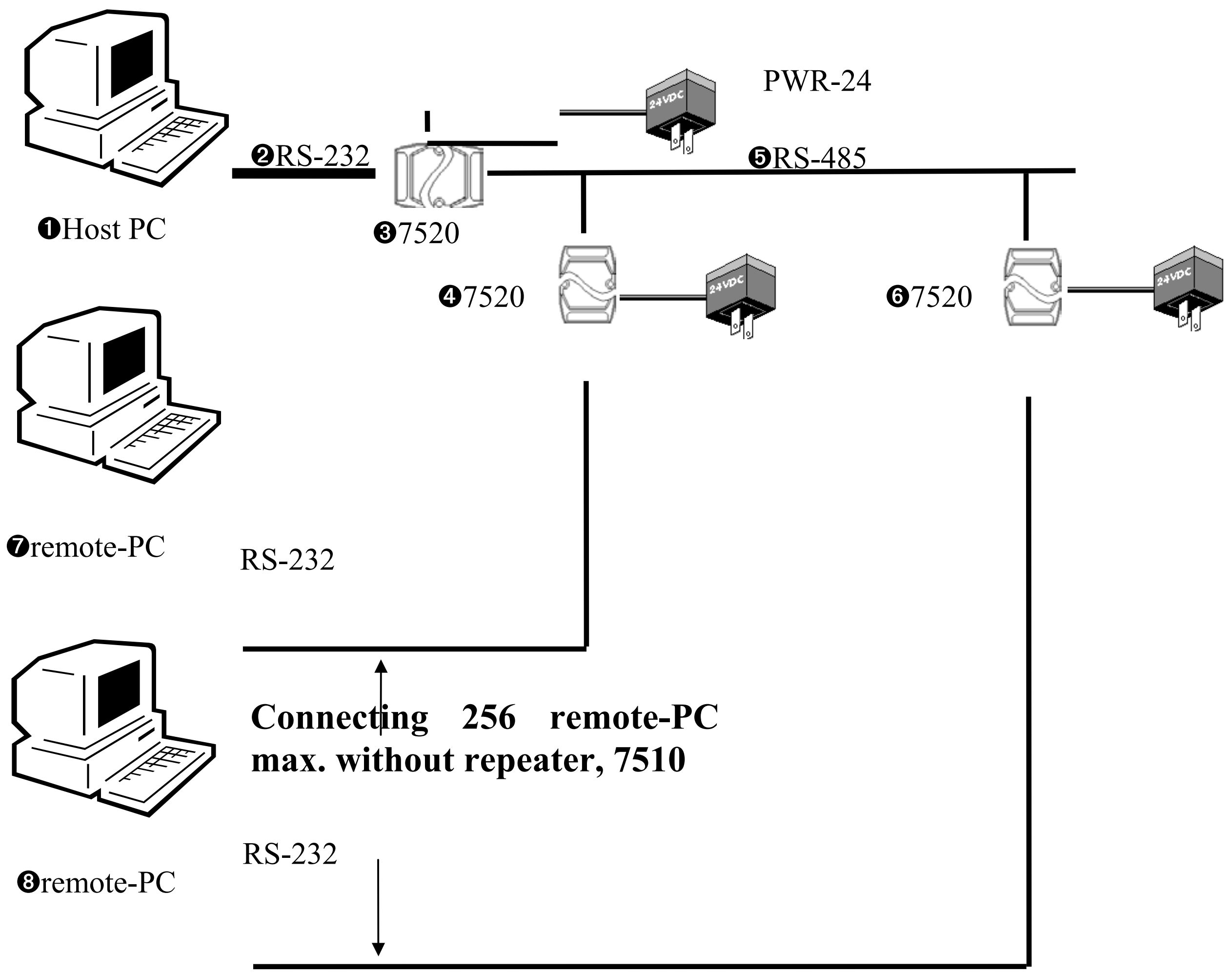


Fig 25

Every remote-PC must has its own address. This address is similar to module address of 7000/8000/87000 series. We call it "remote-PC address". The module address of 7000/8000/87000 is limited to 256, but the remote-PC address is unlimited. The user can connect thousands of PCs in one RS-485 network by using repeater, 7510.

The host-PC can send out command to remote-PC, just like send out command to 7000/8000/87000 modules. The remote-PC will receive command and execute command if the destination address is matched with his local address.

This remote-PC and 7000/8000/87000 series modules can share the same RS-485 network. The host-PC can send out 7000/8000/87000 command and send out PC-networking command at different time. These modules and remote-PC will receive their command respectively. This make the network very low cost, flexible and reliable. This is a unique feature in the world.

4.2 Example

Assume the ❶Host PC has to link 1000 sets of remote PC with the same communication speed and the same RS-485 network, the address field must be 3 byte. The 3-byte address can be from 000 to FFF, totally 1024 different address. The user can design his special command format as desired. For example,

&AAA(string)(chksum)(cr)
& is a delimiter character
AAA=3-byte HEX address, from 000 to FFF
(string)=command string
[chk]=2-character checksum, if checksum disable → no [chk]
(cr)=0x0D

The files in \DCON_Utility_DOS\PC_NET\TC\TEST.EXE are designed to link thousands of remote PC with one HOST PC. The address of host is 0. The address of remote PC are from 1 to FFF, totally 1023 max. The TEST.EXE will read the content of ADDR.DAT. These is only one address data in ADDR.DAT. If this data is 0,.the TEST.EXE will demo as a host PC. If this data is any value other than 0, the TEST.EXE will demo as a remote PC. The demo steps are given as following:

step 1: the user can key-in command in the host PC.
step 2: the host PC send out command to remote PC
step 3: all remote PC receive this host command
step 4: the remote PC with the same address as the destination
address will send result to the RS-485 network
step 5: the host PC receive this result

The steps for host PC to run TEST.EXE are given as following:
step 1: edit ADDR.DAT, key-in 0
step 2: run TEST.EXE
step 3: key-in command string
step 4: repeat step3 until stop
step 5: key-in ‘Q’ command can stop TEST.EXE

The steps for remote PC to run TEST.EXE are given as following:
step 1: edit ADDR.DAT, key-in address (not 0)
step 2: run TEST.EXE
step 3: press any key will stop TEST.EXE

4.2.1 Program Source

The files in \NAP7000\PC_NET\TC are given as below:

1.	UART.H	→ header file
2.	UART.C	→ library file
3.	TEST.C	→ application file
4.	TEST.PRJ	→ project file

These files are very similar to those introduced in Sec. 2.2. Refer to companion floppy disk for program source. Refer to Sec. 2.2 for more information.

is_master()

```
{
int iRet;

for (;;)
   {
   printf("\nCommand="); scanf("%s",cmd);
   SEND_CMD(iComPort,cmd,TIMEOUT,iChksum);
   if ((cmd[0]=='q') || (cmd[0]=='Q')) return;
   iRet=RECEIVE_CMD(iComPort, result, TIMEOUT,iChksum);
   if (iRet==0) printf("Receive=%s",result);
   else if (iRet==1) printf("Receive=com value error (must 1/2/3/4)");
   else if (iRet==2) printf("Receive=Timeout");
   else if (iRet==3) printf("Receive=chksum error");
   }
}
```

```
is_remote()
{
int iRet,addr,i,j,k;
for (;;)
      {
   iRet=IS_DATA(iComPort);
   if (iRet!=0)
      {
      iRet=RECEIVE_CMD(iComPort, result, TIMEOUT,iChksum);
      i=strlen(result);
      result[i-1]=0;
      if (iRet==0) printf("Receive=%s",result);
      else if (iRet==1) printf("Receive=com value error (must 1/2/3/4)");
      else if (iRet==2) printf("Receive=Timeout");
      else if (iRet==3) printf("Receive=chksum error");
      if (result[0] != '&')
         {
         printf(" --> Receive a Error Comand\n");
         }
      else
         {
         i=convert_addr(result[1]);
         j=convert_addr(result[2]);
         k=convert_addr(result[3]);
         addr=i*256+j*16+k;
         If (ADDRESS == addr)
          {
          printf(" --> this is my command\n");
          sprintf(cmd,"Echo from remote_PC_%d",ADDRESS);
          SEND_CMD(iComPort,cmd,TIMEOUT,iChksum);
          }
         }
      }
   if (kbhit()!=0) {getch(); exit(0);}
   }
}
```

```
convert_addr(char c)
{
switch(c)
     {
     case '0' : return(0);
     case '1' : return(1);
     case '2' : return(2);
     case '3' : return(3);
     case '4' : return(4);
     case '5' : return(5);
     case '6' : return(6);
     case '7' : return(7);
     case '8' : return(8);
     case '9' : return(9);
     case 'A' : return(10);
     case 'B' : return(11);
     case 'C' : return(12);
     case 'D' : return(13);
     case 'E' : return(14);
     case 'F' : return(15);
     }
}
```

5 PLC Networking Applications

5.1 Introduction

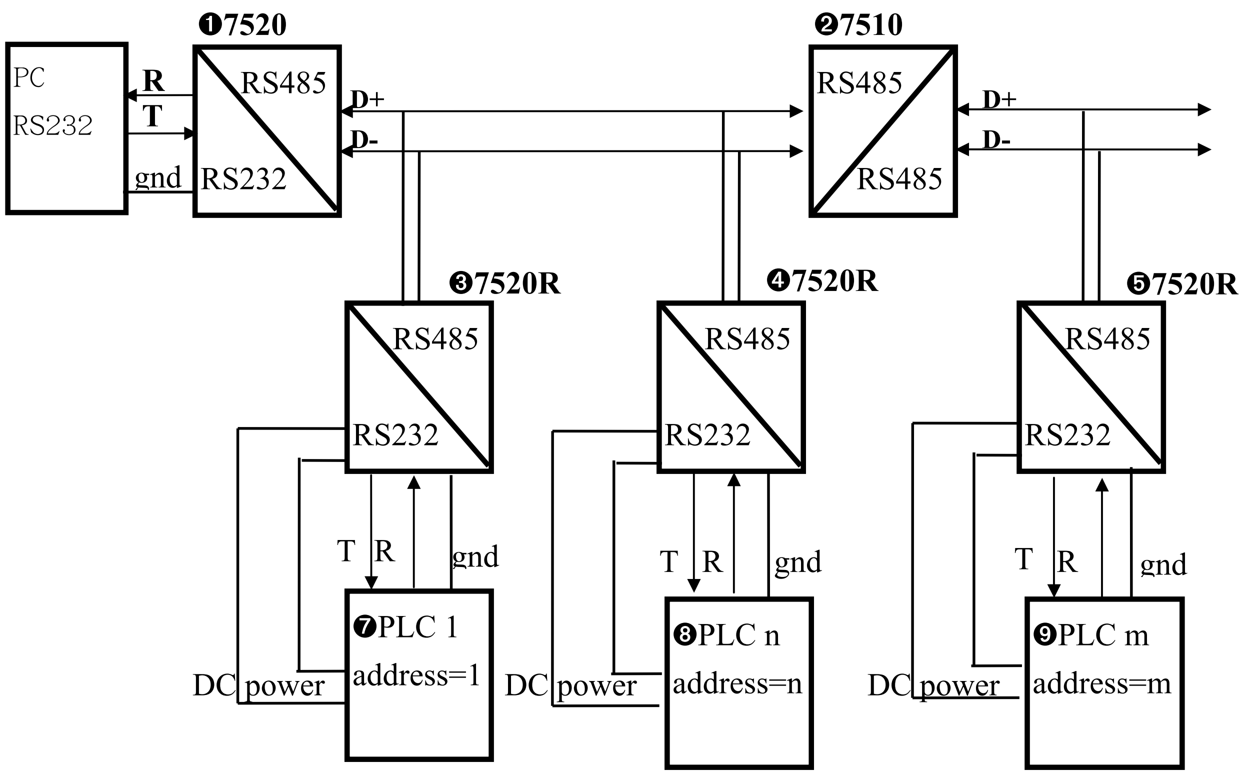


Fig 24

The ❼PLC1, ❽PLCn and ❾PLCm can be used in <u>different baud rate</u> & <u>different configuration</u>. For example,

PLC-1 = 1 start + 7 data + 1 stop = 9-bit/byte
baud rate = 1200
PLC-n = 1 start + 8 data + 1 parity + 1 stop = 11-bit/byte
baud rate = 9600
PLC-m = 1 start + 8 data + 1 parity + 2 stop = 12-bit/byte
baud rate = 115200

In this configuration, the ❼PLC1, ❽PLCn, ❾PLCm provide DC power to ❸7520R, ❹7520R, ❺7520R. This is the cheapest way and can maintain the 3000V high isolation. The user should not replace these 7520R, ❸, ❹, ❺, to 7520. Refer to “7000 Bus Converter User Manual” for more information.

The operation steps are given as following:

step 1: host PC send out RS-232 command to ❶7520.
step 2: ❶7520 convert this command to RS-485 signal.
step 3: ❸7520R, ❹7520R and ❺7520R convert this signal to RS-232 command.
step 4: ❼PLC1, ❽PLCn and ❾PLCm receive this RS-232 command and start to execute.
step 5: the destination PLC send result to 7520R.
step 6: the ❶7520 convert this result signal to RS-232 signal.
step 7: the host PC receive this result and take next action.

5.2 Example

The communication data format of OMRON CQM1 & C200 is the same. The data format is given as following:

OMRON CQM1 = 1 start + 7 data + 1 even parity + 2 stop
= 11-bit/byte
OMRON C200 = 1 start + 7 data + 1 even parity + 2 stop
= 11-bit/byte

This demo will connect three OMRON PLC into one RS-485 network as following:

(1) PCL1: address=00, CQM1, baud rate=9600
(2) PLC2: address=01, CQM1, baud rate=9600
(3) PLC3: address=02, C200, baud rate=9600

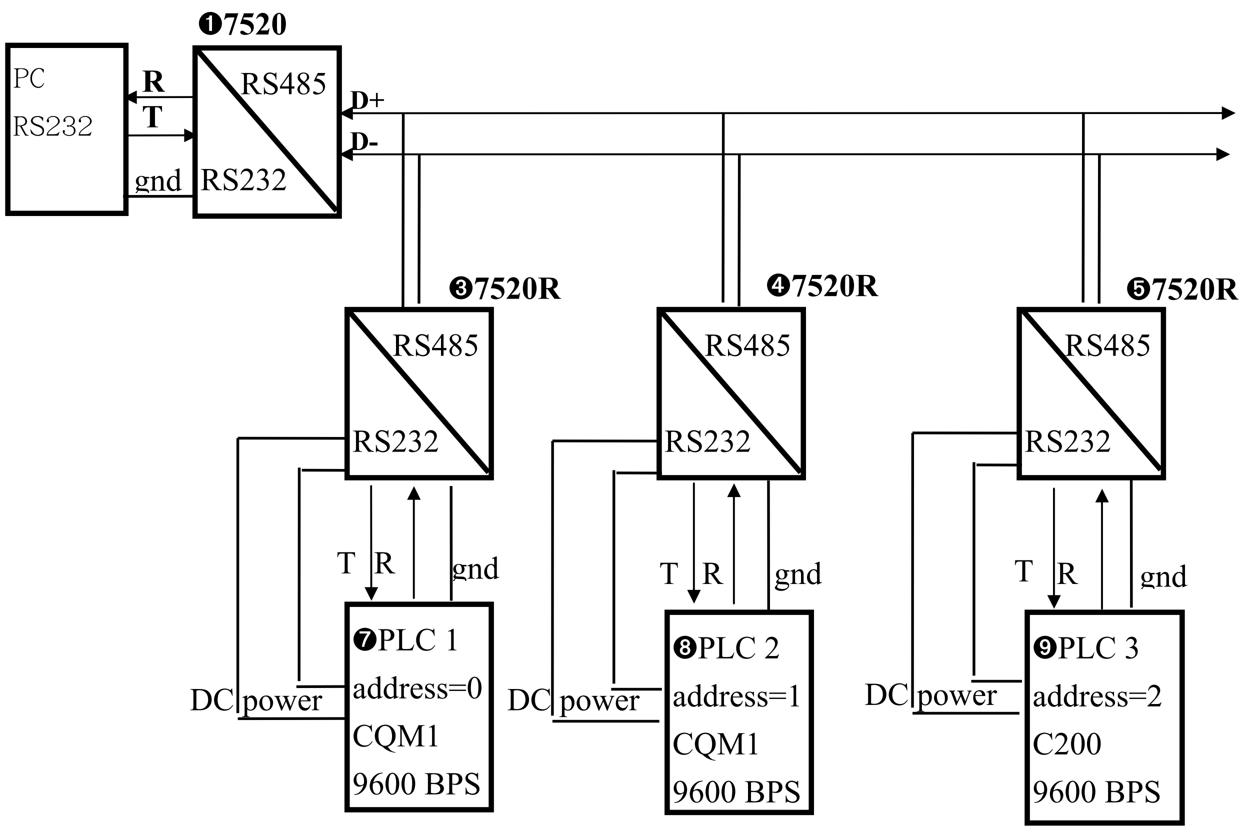


Fig 24

5.2.1 Program Source

The files in \NAP7000S\PLC_NET\TC are given as below:

1.	UART.H	→ header file
2.	UART.C	→ library file
3.	TEST.C	→ application file
4.	TEST.PRJ	→ project file

```
send_command(char cmd[], char result[], int chksum)
{
                                /* ---- function 1 ------- */
                                /* chksum=0 --> no add FCS
                                      =1 --> add   FCS
                                */
int iRet, iLen;
if (chksum==1) add_FCS(cmd);
SEND_CMD(iComPort,cmd,TIMEOUT,0);
iRet=RECEIVE_CMD(iComPort, result, TIMEOUT,0);
iLen=strlen(cmd); cmd[iLen-1]=0;
printf("Send=%s",cmd);
if (chksum==1)
  {
  if (iRet==0) printf(" --> Receive=%s",result);
  else if (iRet==1) printf(" --> Receive=com value error (must 1/2/3/4)");
  else if (iRet==2) printf(" --> Receive=Timeout");
  else if (iRet==3) printf(" --> Receive=chksum error");
  }
printf("\n");
}
```

```
add_FCS(char *td)
{
int len, i;
char aa[3];
unsigned int a;
len = strlen(td);
a = td[0];
for (i=1; i<len; i++) a ^= td[i];
itoa(a, aa, 16);
if (strlen(aa)==1) strcat(td, "0");
strcat(td, aa);
strupr(td);
strcat(td, "*\x0d");
}

/* ---- function 2 -------------------------------------------------- */

to_plc1()
{
strcpy(cCmd1,"@00SC02");       /* change to MONITOR mode
*/
send_command(cCmd1,cCmd2,1);
strcpy(cCmd1,"@00WD00040001");  /* write DM_0004 */
send_command(cCmd1,cCmd2,1);
strcpy(cCmd1,"@00WD00050002");  /* write DM_0005 */
send_command(cCmd1,cCmd2,1);
strcpy(cCmd1,"@00SC03");    /* change to RUN mode */
send_command(cCmd1,cCmd2,1);
printf("---------------------------------------------------\n");
strcpy(cCmd1,"@00RD00040001");  /* read DM_0004 */
send_command(cCmd1,cCmd2,1);
strcpy(cCmd1,"@00RD00050001");  /* read DM_0005 */
send_command(cCmd1,cCmd2,1);
strcpy(cCmd1,"@00RD00070001");  /* read DM_0007 */
send_command(cCmd1,cCmd2,1);
}
```

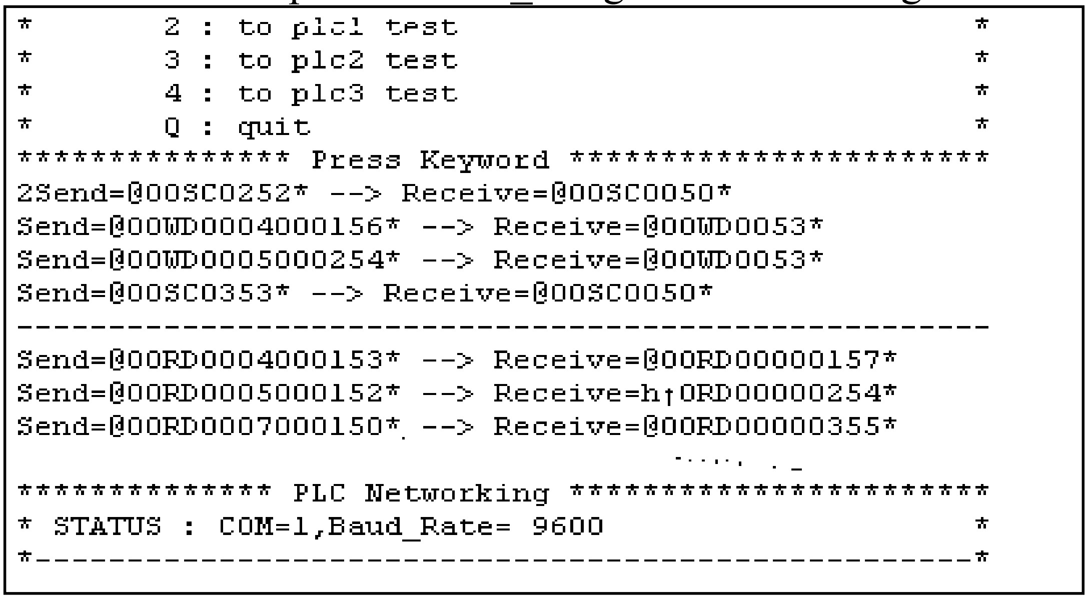
5.2.2 PLC Command

- @**(cr): power-on initialization command
- @AATS(test string)(FCS)(cr): test command
- @AARDSSSSNNNN(FCS)(cr): read DM command
- @AAWDSSSSNNNN(FCS)(cr): write DM command
- @AASC02(FCS)(cr): change to MONITOR mode
- @AASC03(FCS)(cr): change to RUN mode

5.2.3 PC Test Function

- function_0: PC COM port initialization
- Refer to Sec. 2.1.1
- function_1: send out TEST command
- Refer to Sec. 2.1.3
- function_2: send command to PLC_1
- change DM_0004 & DM_005, read back DM_0004, DM_0005, DM_0007
- function_3: send command to PLC_2
- same as function_2
- function_4: send command to PLC_3
- same as function_2

The screen dump of function_2 is given as following:

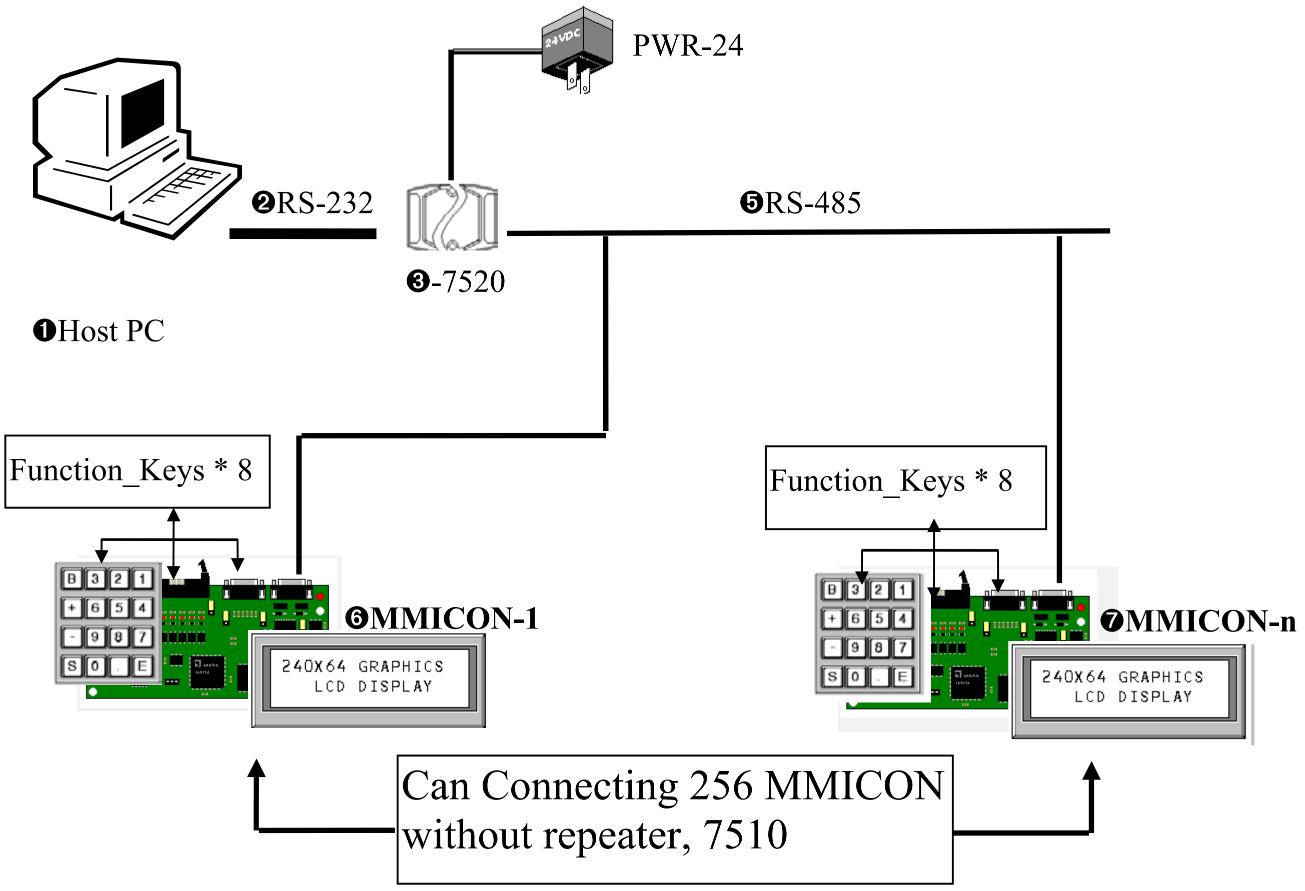


```
*       2 : to plc1 test                                  *
*       3 : to plc2 test                                  *
*       4 : to plc3 test                                  *
*       Q : quit                                          *
*************** Press Keyword ************************
2Send=@00SC0252* --> Receive=@00SC0050*
Send=@00WD0004000156* --> Receive=@00WD0053*
Send=@00WD0005000254* --> Receive=@00WD0053*
Send=@00SC0353* --> Receive=@00SC0050*
-----------------------------------------------------
Send=@00RD0004000153* --> Receive=@00RD00000157*
Send=@00RD0005000152* --> Receive=h↑0RD00000254*
Send=@00RD0007000150* --> Receive=@00RD00000355*

************** PLC Networking ************************
* STATUS : COM=1,Baud_Rate= 9600                          *
*-------------------------------------------------------*
```

6 MMICON Applications

6.1 Introduction


PWR-24
24VDC
❷RS-232
❸-7520
❺RS-485
❶Host PC
Function_Keys * 8
❻MMICON-1
240X64 GRAPHICS LCD DISPLAY
Function_Keys * 8
❼MMICON-n
240X64 GRAPHICS LCD DISPLAY
Can Connecting 256 MMICON without repeater, 7510

Refer to "MMIDOS Software User Manual" for demo program.

MMICON command sets.

command syntax	response syntax	documentation
$AAPDD	!AA	change display page,AA=MMICON address,DD=page num
$00P00	!01	change to page_0 (default pin=0 → AA=0)
$00P01	!01	change to page_1 (default pin=0 → AA=0)
$AATVHHStr	!AA	**show string**, AA=MMICON address,V=0-7, HH=0-14(hex)
		Str=string to be shown on LCD
$00T002Hello	!01	show **Hello** in row=0, column=2
$00T310Test	!01	show **Test** in row=3, column=0x10
$AAK	!AAVKeys	read 4*4 keyboard,if key buffer overflow then V=1 else V=0
		Keys=keys pressed code, refer to “MMICON user manual”
$00K	!010	for keycode details
$00K	!01019010314	no keys pressed
$00K	!01002	[19] [01] [03] [14] total 4 keys are pressed
		[02] total 1 key is pressed
%AANNTTBB00	!AA	**change configuration**
AA=current addr		
NN=new addr		
TT=mode number		refer to Sec. 7.1 for operating mode
= 00 → mode 0		digital I/O interface mode
= 01 → mode 1		PC RS232/RS485 interface mode
= 02 → mode 2		PC RS232 interface mode
= 03 → mode 3		PLC RS232 mode
BB=baudrate		**RS232/RS485 baudrate**
= 03 → 1200		
= 04 → 2400		
= 05 → 4800		
= 06 → 9600		
= 07 → 19200		
%0001010600	!01	change to PC RS232/RS485 interface mode
%0001030600	!01	change to PLC RS232 interface mode
%0001020600	!01	change to PC RS232 interface mode
$AA2	!AATTBBFF	read current configuration
$002	!01030600	mode-3, PLC RS232 interface mode
$002	!01010600	mode-1, PC RS232/RS485 interface mode
$AAM	!AAMMICON	read module name
$00M	!01MMICON	
$AAF	!AAA?.?	read firmware version number
$00F	!01A2.3	software version=2.3

7 5860 Applications

7.1 5860/5/7/12 Specifications

(1) CPU CARD: SSC-5x86HVGA

(2) RACK:	PR105	for 5860/5
	PAC-70H	for 5860/7
	RACK-300H	for 5860/12

(3) RAM: 4M
(4) PROMDISK: 2M (with DOS & 5860 utility)
(5) CPU: AMD 5x86-133
(6) FLOPPY DRIVER: 1.44M*1
(7) ISA-7520R

(8) EMPTY SLOT:	3	for 5860/5
	5	for 5860/7
	10	for 5860/12

(9) SOFTWARE: NAPDOS9
(10) OPTIONS: I/O boards

1. ISO_AD32L/H
2. ISO_DA8/16
3. ISO_P32C32
4. ISO_P64
5. ISO_C64
6. P8R8DIO
7. P16R16DIO
8. ISO-813
9. More

7.2 5860 features

(1) PC based embedded controller, 2M solid state electronic disk on board
(2) High density, high speed I/O, isolated A/D, D/A, D/I, D/O boards
(3) Accept command from RS-485 network, command sets are similar to 7000 series
(4) User can write his application program by any programming language

7.3 5860 Software Environment

The 5860 software is a TSR program which will automatically run after power-on. This TSR program will accept command from RS-485 network and execute this command if the command destination address is match with 5860's device address. Therefore the user can send command to 5860 very similar to 7000 series modules.

The partial command sets are given as following:

- &AA0B → information request of board_B
- &AA1B(data) → read from D/I of board_B
- &AA2B(data) → write to D/O of board_B
- &AA3BNNCC → set A/D configure_CC of channel_NN, board_B
- &AA4BNN(data) → read from A/D channel_NN data of board_B
- &AA5BNN(data) → write to D/A chanel_NN of board_B
- &AAA(data) → write to user program
- &AABNN → read from user program

The TSR will put all I/O states in the PC memory or file. The user can write his application using QB or C under DOS. The user does not have to write I/O driver, he can access these I/O states through PC memory or PC file. If there is no user program, the 5860 will function very similar to 7000. The 5860 command sets are also very similar to 7000.