Appendix K Application Notes

1. There is a way to monitor the line number that is currently being executed by each of the tasks. A 'line pointer" is located at the addresses $OA (MSB) and $OB (LSB) for the first task. This pointer 'points' to a location in RAM, containing the line number. The pointer of each additional task is located $100 higher.

	MSB	LSB
Task 1 Linepointer	$0A	$0B
Task 2 Linepointer	$10A	$10B
Task 3 Linepointer	$20A	$208
Task 4 Linepointer	$30A	$30B

If you attempt to monitor a linepointer for a task within that task, you will always return the same value.

This function is similar to a BASIC TRACE statement (TRON /TROFF). Unlike the TRON/TROFF statement (which shows every statement that is executed), the next example program will only sample, every few timer ticks, the current line pointer in TASK 2. However, useful in displaying the linenumber of a statement that is repeatedly executed (hung statement, or in an infinite loop, for example).

Example: (monitor the linenumber of the second task)

```
10   TASK 100
20   POKE($F6,1)                          Set unsigned calculations
30   A=256*PEEK($10A)+PEEK($10B)          Get pointer to second task
40   B=256*PEEK(A)+PEEK(A+1)              Get line number
50   PRINT B                              Print line number
60   GOT0 30
100  REM Second Task
```

2. You can restart the program from within the program. Certain precautions must be taken to prevent the watchdog timer from tripping. This requires POKEing some assembly code that disables interrupts before jumping to the 6809 restart vector.

```
10   POKE ($C3F1,$1A)                     Disable Interrupts
20   POKE ($C3F2,$50)                     Disable Interrupts
30   POKE ($C3F3,$6E)                     Jump indirect to Restart
40   POKE ($C3F4,$9F)                          "
50   POKE ($C3F5,$FF)                          "
60   POKE ($C3F6,$FE)                          .

xxxx EXEC $C3F1                           Restart program
```

If only one task is programmed, it is not necessary to disable interrupts, In that case it is only necessary to jump to the 6809 Restart Vector. After PEEKing the addresses $FFFE (MSB) and $FFFF (LSB) you will have the restart vector for the microprocessor. For convenience (since additional tasks may be programmed), it is recommended that the interrupts are disabled.

3. A BASIC PROM can be modified so that the interpreter ignores a <BREAK> character. The <BREAK> is normally used to stop a running program. However, it may be desirable to stop a running program to examine variables or to make changes. This can be done with the END statement. The END can be made conditional on certain items (such as entering a password):

```
100     TASK 10000                          :REM point to concurrent task
110     SUSPEND 10                          :REM blink LED 1
120     LEDON
130     SUSPEND 10
140     LEDOFF
150     GOT0 110                            :REM end of blink routine
10000 REM ----- TASK #2 ----
10010 REM
10020 PORT A 9600,N,8,1                     :REM Point to PORT A (plug terminal here)
10030 IF LOC > 0 THEN GOT0 10100            :REM any characters in buffer?
10035  LED2ON                               :REM light LED 2
10040 SUSPEND 50                            :REM no, wait one second
10050 GOT0 10030                            :REM and check the input buffer again
10100 REM ---- character in buffer ----
10105  LED2OFF                              :REM extinguish when processing password
10110 SUSPEND 250                           :REM window of opportunity is 5 seconds
10120 IF LOC > 5 THEN GOT0 10500            :REM 5 character plus <CR>
10130 GET A$,LOC                            :REM too few, purge buffer
10500 REM --- minimum number of characters received ----
10510 GET A$,LOC                            :REM fetch those characters
10520  V=INSTR(A$,"EJECT")                  :REM the password is EJECT
10530 IF V>0 THEN END                       :REM yes, correct password entered
10540 GOT0 10030                            :REM no, test for more data
```

Many times PORT A will already be in use by a terminal. In that case, the Password code must be 'interleaved' with the normal terminal servicing code.

```
100     PORT A                              :REM look at port A
110     IF LOC=0 THEN SUSPEND 10:GOTO 110 :REM wait for characters
120     REM Wait for two characters to be received
130     IF LOC<2 THEN SUSPEND 1 :GOTO 130
140     GET A$,2                            :REM get those two characters
150     REM ----  Set up a test loop ----
160     IF A$(1)="1" THEN GOSUB 1000        :REM if '1' entered, do this
170     IF A$(1)="2" THEN GOSUB 2000        :REM if '2' entered, do this
180     IF A$(1)="3" THEN GOSUB 3000        :REM if "3" entered, do this
190     IF A$(1)= 3 THEN GOSUB 4000         :REM if <CTRL>-C, process password
200     GOT0 160                            :REM continue looping

4000    REM  --- Process a password request ----
4010    TM=0                                :REM zero out timer
4015    PRINT 'PASSWORD - ";                :REM prompt for password entry
4020    IF LOC = 8 THEN GOT0 4050           :REM wait for 8 character password
4030    IF TM > 200 THEN RETURN             :REM times out after 4 seconds
4040    GOT0 4020                           :REM keep looking for 8 characters
4050    GET A$,8                            :REM fetch these 8 characters
4060    IF A$="PaSsWoRd" THEN PORT A:END    :REM stop, password correct
4070    RETURN                              :REM password incorrect, continue
```

4. The module can be programmed to attempt an EPROM restart following a watchdog timer trip. This feature may be useful where the module is used as a Modbus or IAN translator, since a loss of communications with the module may present a problem. Of course, pressing the reset button will reset the module, but perhaps the button is not easily accessible.

> Warning!
>
> Under no circumstances should this feature be used if the 1799 module is performing control (manipulating outputs). A watchdog trip may indicate an internal failure resulting in unexpected results. The user is cautioned against using this feature for any application where a failure of the 1799 module may result in equipment damage or present a safety hazard to personnel.

To accomplish this, read the NMI (non maskable interrupt) vector located at addresses $7FFC (MSB) and $7FFD (LSB) (Note: $7FFC and $7FFD locations in PROM correspond to CPU memory locations $FFFC and $FFFD). This will point to a location in the EPROM. Starting at that location change the next 5 bytes to the following new values:

+0	= $1A	Disable Interrupts
+1	= $50	
+2	= $9F	Jump Indirect to restart
+3	= $FF	
+4	= $FE	

In version 1.5 of BASIC, the NMI vector in the PROM points to memory address $E775. Since a PROM address is $8000 less than a CPU address, we subtract $8000 from $E775. The result ($6775) is where we patch in the new code:

	New	Old (V1.5)
$6775 =	$1A	$86
$6776 =	$50	$10
$6777 =	$9F	$C6
$6778 =	$FF	$08
$6779 =	$FE	$FD

Appendix L Modbus Emulation V3.3

An optional Modbus slave emulation package can be added to the BASIC interpreter. The Modbus runs independently of the BASIC. For information on Modbus Master capability refer to the MBRCV (page 52) and MBXMT (page 53) functions built in to the BASIC interpreter.

> Warning
>
> The Modbus interpreter supplied with the NCMZ-1799 module will NOT be robust enough for industrial applications unless the final program is burned into the Executive EPROM and the 'Break Detect' function disabled. If these actions are not taken, the Modbus executive will be susceptible to electrical noise, and may stop running. It would then be necessary to connect a terminal and issue a RUN command or (possibly) reload the program.

This document describes the MODBUS SLAVE EMULATION communications package that can optionalfy be supplied with this module. BASIC is the resident software for the module. The emulation is invoked by a few BASIC Executive statements.

MODBUS messages of up to 125 words are supported. The functions implemented are 1, 2, 3, 4, 5, 6, 7, 8 and 16 are described in more detail in section L.6 of this appendix.

The emulation software will convert MODBUS messages, coming in on one dedicated NCMZ port from the MODBUS LINK, into standard Westinghouse 6-byte protocol PROGRAM LOADER style messages relayed to the PLC on a second dedicated port. Since the NCMZ to PLC connection is dedicated with regards to which NCMZ port is used, two separate forms of the MODBUS SLAVE EMULATION have been written:

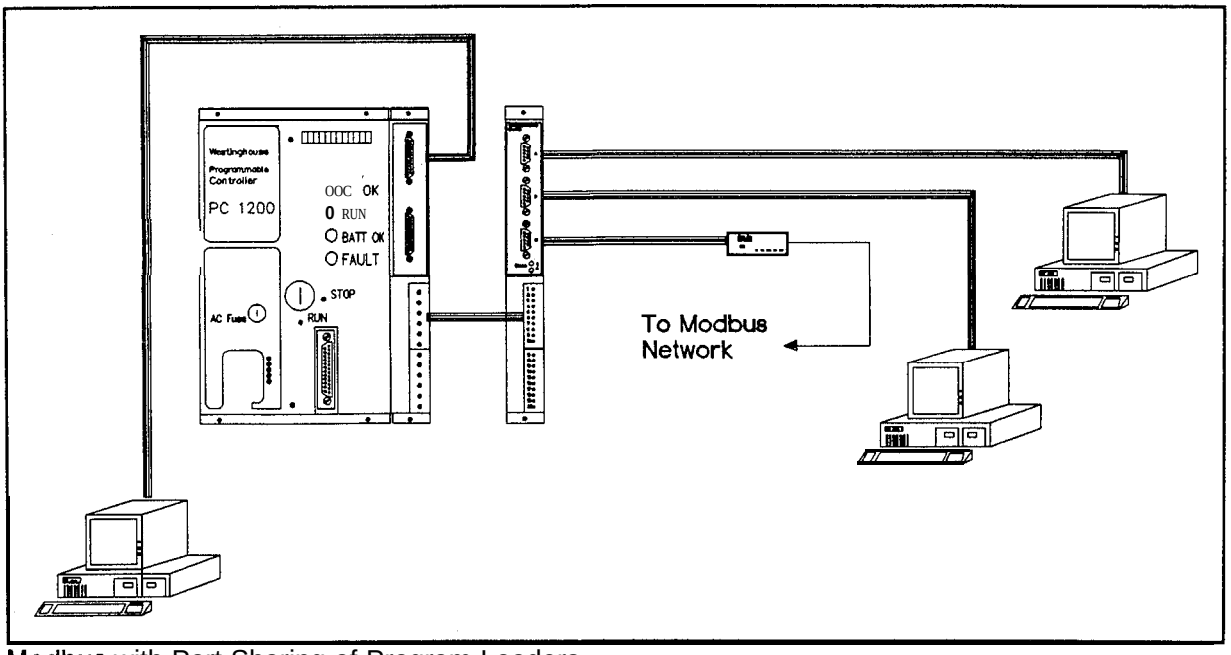
SP1799 V3.3C	NCMZ Port B - MODBUS LINK interface NCMZ port C - PLC interface
SP1799 V3.3D	NCMZ Port C - MODBUS LINK interface NCMZ Port D - PLC interface

In addition to the MODBUS emulation software, several port sharing routines can be started as part of either option. One routine will designate a Program Loader Interface on the NCMZ module to time share the NCMZ/PLC interface. The Program Loader port supports PROGRAM LOADER Block Read and Block Write messages of up to 64 words, as well as all conventional 6-byte messages.

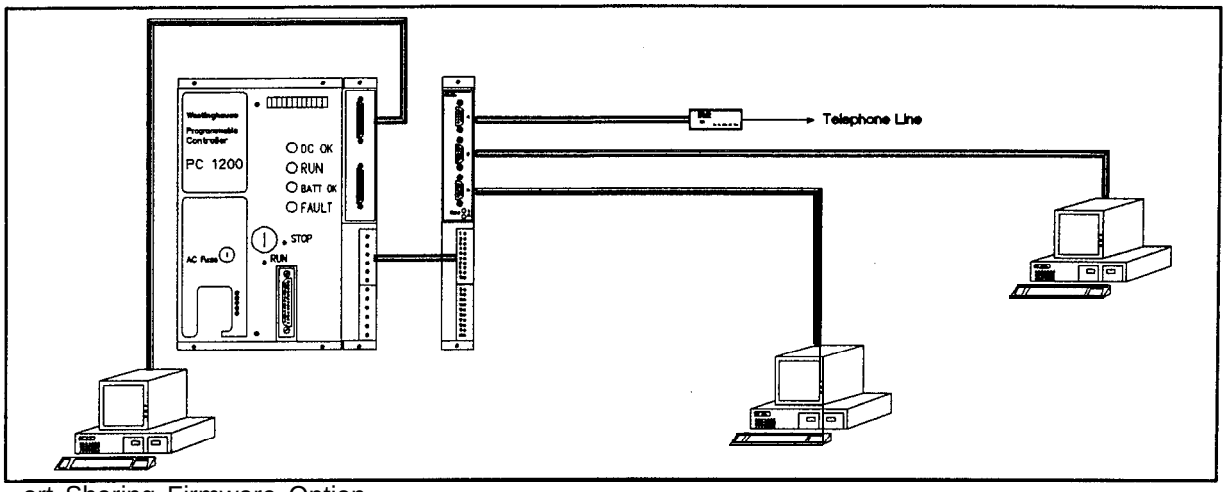
The drawing shown on the next page ("Modbus with Port Sharing of Program Loaders) demonstrates possible ways of connecting field devices to the 1799 module. Notice that the connection between the 1799 and the PC1200 is via the RS-485 twisted pair. Therefore the Modbus PROM option is V3.3D.

Another drawing (Port Sharing Firmware Option) shows the 1799 module with Modbus V3.3D, but all ports are designated as Westinghouse Program Loader ports. No ports are configured as Modbus ports.

In addition to the port share and Modbus capability, two more routines are available to allow serial communications with the Programmable Controller from your BASIC program. These routines use the



Modbus with Port Sharing of Program Loaders



ort Sharing Firmware Option

same 1799 to PLC link that the port share and Modbus functions use. Therefore this port must be 'claimed' by the BASIC program when needed. The link should be 'released' after the BASIC program is finished so that the port share and Modbus routines can process any messages they may have been stored in the interim.

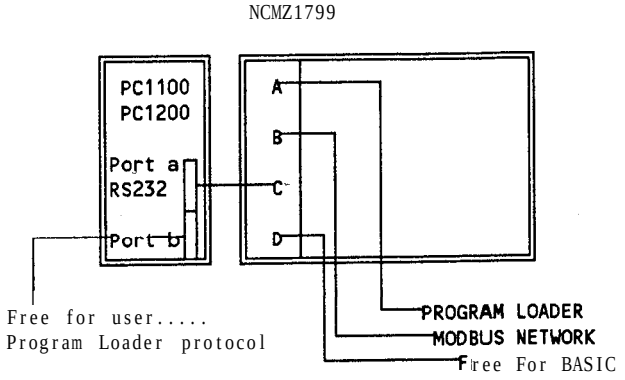
The first routine will claim the NCMZ/PLC interface port using an EXEC $BEOO. From this moment on, messages from either the MODBUS LINK or PROGRAM LOADER interface will be saved until the NCMZ/PLC interface is free again, The PLC interface is released by the second routine (EXEC $BF00).

These two routines (claim and release) are only useful when the BASIC version 1.4 or greater is implemented with the Block Read and Block Write functions.

L.l Hardware Installation

The module must be plugged into an unused slot of the PLC's I/O rack. The host PLC can be a PC1 100 or 1200 processor. Proper functioning of the module is independent of its position in the I/O rack, but for simple cable arrangements the first slot in the main rack is preferred. For more information on the hardware setup of this module, refer to the NCMZ-1799 Instruction Leaflet and the PC1 100/1200 Systems Manual NLAM-B206 section 3-20,21 on networking and serial port definition.

L.2 SP1799 V3.3C Option



NCMZ Port Definition for SP1799 V3.3C

Port A — For use by the BASIC program or optional used as a time shared PROGRAM LOADER port.

Port B — Designated as the MODBUS NETWORK LINK or PROGRAM LOADER interface,

Port c — The dedicated NCMZ/PLC interface for data access with port A of the PC1 100/1200. The RS232 interface of PLC's Port B can be used instead if it is left in the single point mode.

L.2.1 Cascading Port Sharing Modules

Version C of the Modbus 3.3 firmware permits cascading 1799 modules. This would allow adding even more serial ports to the PLC. Be aware, however, that this will result in a reduced throughput to the PLC.

NOTE

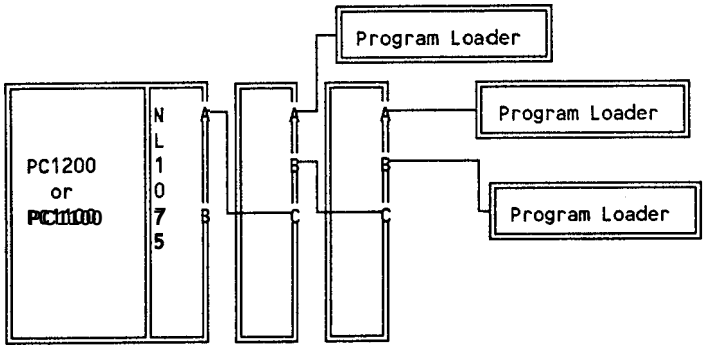
Although the port sharing software supports multiple program loaders on each port, only one of those program loaders may be displaying ladder in the Monitor Mode. If more than one program loader attempts to display ladder in the Monitor Mode, the power flow display will only be correct on the last program loader to display a rung.

Operation of the logic in the controller is not, however, affected.

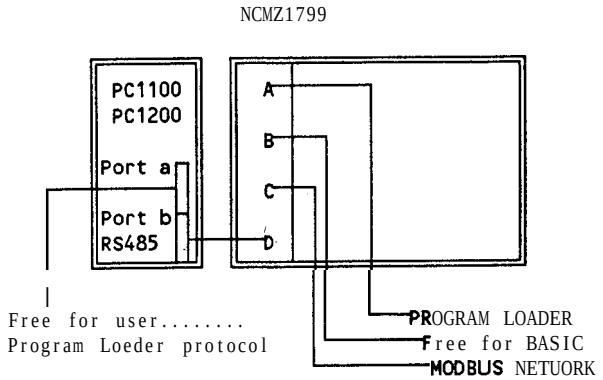
Warning

Although the port sharing program can be added to existing BASIC program, the port sharing routine will not be robust enough for industrial applications unless this routine is burned into the executive PROM and the 'Break Detect' function disabled. Failure to do this could result in a program that stops running

Cascading 1799 Modules



L.3 SP1799 V3.3D Option



NCMZ Port Definition for SP1799 V3.3D

Port A — PROGRAM LOADER interface or may be left free for the user.

Port B — Can be chosen to be a PROGRAM LOADER interface, however since the main PLC-program loader interface is not used it may be redundant and port B can be used for other purposes in your BASIC application.

Port c — Designated as the MODBUS NETWORK or PROGRAM LOADER interface.

Port D — The SP1799 V3.3D package uses the modules port D (RS485) interface as a dedicated data access with port B of the PLC. Port A of the PLC is not available in RS485. Special considerations must be taken to connect the module and PLC:

a) Mode switch on PLC-processor board must be left open (processor to single program-loader position).

b) Interconnect points 13 (CTS Port B) and 22 (DTR Port B) on the 25-pin communications interface of the host PLC. If a PC1200 is used, this step is not necessary. However, a Configure Port (CP) function must be used. Set the 16th bit of the CONFIGURE register high (disables CTS testing). Refer to Appendix J (Troubleshooting) for more information.

c) Connect pin 17 of the 25-pin communications interface of the host PLC with pin 2 of the front edge connector of the module (TXRXA).

d) Connect pin 15 of the 25-pin communications interface of the host PLC with pin 3 of the front edge connector of the module (TXRXB).

e) Connect pin 11 of the 25-pin communications interface of the host PLC with pin 4 of the front edge connector of the module (SYNCA).

f) Connect pin 24 of the 25-pin communications interface of the host PLC with pin 5 of the front edge connector of the module (SYNCB).

L.4 Resident Software and Modbus Emulation Activation

The BASIC interpreter is the resident software of the module. The MODBUS emulation software is added to the resident EPROM as an option and runs as a background task from an executive call. To invoke the MODBUS emulation, a small basic program is required. The minimum program that is needed, is given below.

```
10 AUTO ON                    This will automatically start the program after a power cycle
20 PORT B 9600,N,8,1          Sets MODBUS parity and baud rate
30 PORT C 9600,0,8,2          Sets PLC port parity and baud rate
40 EXEC $B000                 Start MODBUS emulation software
90 SUSPEND 100:GOTO 90        Endless loop
```

Once the emulation is running, the NCMZ to PLC interface is dedicated and cannot be redefined. The interface link can be shared by **either a program** loader or by the BASIC (V1.4 or greater) NCMZ user program.

On power up, the module will first read the parameter table from the PLC (locations 8200 - 821F Hex), This table contains data about number of discrete I/O's, the number of register I/O and Holding Register usage and it is used to check the validity of addresses coming from the MODBUS master, In addition the contents of HR5 is read, which holds the RTU number of this Modbus slave. During this start-up procedure, status LED 2 will flash. Upon completion, the LED will go steady.

In the BASIC program from line 50 and following, an additional BASIC program may be entered. In the SP1799 V3.3C option, only Port A and Port D are available to BASIC, since the other ports are assigned to the MODBUS-emulation. Like wise the SP1799 V3.3D option has Port A and Port B available for BASIC.

L.5 Port Sharing of NCMZ to PLC Serial Interface Link

Since the MODBUS slave emulation software uses one of the PLC ports it is often desirable to time share the port with another device connected to the Module. To this end, an executive call from BASIC can designate Port A as a programmer port for the host PLC. If a 6-byte program loader message is then received, this message will be sent via Port C to the PLC and the answer is returned to Port A via a port sharing technique. The earlier described program must have the following added:

```
50 SUSPEND 500          Allows interrupting program (using <BREAK>) up to 10 seconds after program
                        startup.
60 PORTA 9600,0,8,2     Sets the programmer port baud rate and parity
70 EXEC SBA00           Start the PROGRAM LOADER port
```

> Note
>
> You must program a delay of at least 2 seconds (SUSPEND 100) following the EXEC $8000 statement before processing an EXEC $BA00.

To activate more than one serial port as a program loader port (2 maximum with V3.3C and 3 maximum with V3.3D), program additional EXEC $BA00 statements.

V3.3C

```
72      PORT B 9600,0,8,2   :REM Set baud and data format for PORT B
74      EXEC SBA00          :REM Activate a second program loader port on Port B
```

V3.3D

```
72      PORT B 9600,0,8,2   :REM Set baud and data format for PORT B
74      EXEC SBA00          :REM Activate a second program Loader port on Port B
76      PORT C 9600,0,8,2   :REM Set baud and data format for PORT C
78      EXEC SBA00          :REM Activate a third progrem loader port on Port B (override Modbus)
```

With software version BASIC VI.4 (and later) it is possible to communicate with the host PLC over a serial link. In this case using SP1799 V3.3C, the serial link is already established by the connection port-C <--> Program Loader Interface. It is used by both port B (MODBUS) and port A (Program Loader). If, for any reason, your BASIC application also requires access to the PLC over a serial link, this communication link has to be temporarily claimed by BASIC, and released afterwards.

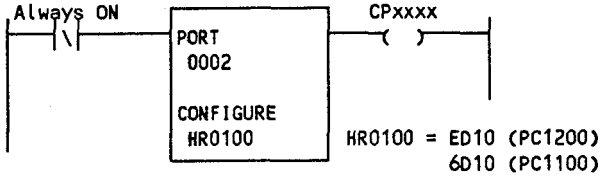
The two routines EXEC $BE00 (CLAIM) and EXEC $BFOO (RELEASE) are available in the form of EXEC statements. If port C is claimed by BASIC, incoming messages from MODBUS or PROGRAM LOADER are buffered. If Port C is in use at the moment of CLAIM, the Basic program will wait until port C is available, and only then return execution to the task. In the example given below the Block Read and Block Write functions are used in a small additional BASIC program.

```
 90 REM                    communicate within BASIC over same Port C
100 DIM S(100)             initialize buffer
110 PORT C                 communicate via Port C
120 E=0                    reset error flag
130 EXEC $BE00             claim Port C
140 BREAD 10,60,S(1),E     read HR(10) to HR(69) into S(1) to S(69)
150 EXEC SBFOO             release Port C
160 IF E<>0 GOT0 120       test if block read is successful
170 EXEC $BE00
180 BWRIT 100,60,S(1),E    write registers to HR(100) to HR(160)
190 EXEC $BF00
200 GOT0 110
```

If you just wish to use this module as a port sharing module (no Modbus support), then a simplified program can be used:

```
100 PORT D 9600,0,8,2      set baud on PORT D to 9600, parity to odd, data bits to 8 and 2 stop bits
110 PORT A 9600,0,8,2
120 EXEC SBA00             activate PORT A as a shared program loader port
130 PORT B 9600,0,8,2
140 EXEC SBA00             activate PORT B as a shared program loader port
150 PORT C 9600,0,8,2
160 EXEC SBA00             activate PORT C as a shared program loader port
170 SUSPEND 100:GOTO 170   loop here
```

Program this statement into the PC1200 (or 1100):



Make sure the MODE switch on the PC1000 is set to 'SINGLE POINT' and that the RS-485 termination resistors switches are set to ALL CLOSED.

Make sure that there is no UNIT ADDRESS (UA) function in memory.

Make sure that the 4 wires (TxRx A, TxRx B, Sync A and Sync B) are connecting the NL-10758 to the NCMZ-1799.

Cycle power to the PLC after all programming or DIP changes have been made.

L.6 Modbus Overview and Emulation Details

The Modbus communications network was designed to provide a simple means of connecting slave controllers to a master controller for the purpose of distributed control and data acquisition. The simplicity of the protocol has made it a popular choice among many control and instrumentation vendors seeking a 'common" language.

The Modbus protocol consists of a slave address, function code, data, and error checking. The master controller issues a command to which the appropriate slave responds.

The following describes the message parameters:

Address
The address field contains the address of a specific slave (l-255) as assigned by the user. Each slave must be assigned a unique address. Only the addressed slave will respond to a message. The response message contains the address of the slave which issued it.

A broadcast message uses a slave address of '0'. All slaves on the network interpret the instruction and take action, but do not issue a response message. Only function codes 5, 6 and 16 can be used with this feature.

Function Code
This field defines the action to be performed. Implemented Functions Codes are 1, 2, 3, 4, 5, 6, 7, 8 and 16. The following table lists the function codes supported by the NCMZ-1799 module and how they are interpreted.

01 Read Coil Status
Obtains the current status (On/Off) of a group of logic coils. Unrequested coils in the response message are zeroed.

02 Read Input Status
Obtains the current status (On/Off) of a group of discrete inputs.

03 Read Holding Registers
Obtains the current value in one or more Holding Registers. A maximum of 125 registers is allowed per request.

04 Read Input Registers
Obtains current value in one or more input registers.

05 Modify Coil Status
Sets logic coil to a state of On or Off. Note the change is made to the output status table and NOT THE FORCE TABLE in effect overriding the force table. Although this function will write to the coil regardless of the status of the force bit, the coil will be immediately overridden by the ladder program. If you wish to actually override the force state, you will need to locate the address of the force bit in the PLC memory (refer to the Westinghouse PLC Communications Manual).

06 Modify Register Contents
Write information to one register. The Module does not limit writes to below HRRU, therefore the user should use extreme caution when writing to any memory location other than Holding Registers.

07 Read Exception Status
The low order byte of HR6 is read. The contents of HR6 may be set by the Ladder Program.

08 Loop Back Diagnostic
Diagnostic test message which can be used to test communications. The response message is identical to received message. The test message must be the full length of a normal command (8 bytes)

16 Preset Multiple Registers
Information can be written to Multiple Holding Registers (up to 125 at a time) with one command.

Data
The data field contains information needed to perform the specific function. This may be values, addresses, or limits, For example, the function code Read Coils (01), requires data of starting coil number and quantity.

Error Check
This field contains the error checksum which is used to determine if a transmission has been received correctly. The Cyclical Redundancy Check (CRC) is used for the RTU format used in the 1.

The package emulates a MODBUS slave in RTU-mode, ASCII mode is not available. The unit address is defined by HR(5) in the PLC. Baud rate and framing have to be set by the BASIC PORT statement, Port definitions are given in section 2.0 of this Manual.

Status LED LED 2 is used as diagnostic indicator

Cont. on -	system OK
Flashing fast -	start-up sequence
Flashing slow -	errors data received from network

Special note

Although the emulation supports messages up to 125 points from the MODBUS network, it should be noted that the 6-byte protocol only supports messages up to 64 words. In case of messages above 64 points, the incoming MODBUS message is split up into two PLC- messages. The response message consequently contains data, that is derived from the PLC in different ladder scans with a resulting slower response time.

ExceptionResponse

Exception error codes are generated by the slave if the master requests an invalid point, quantity or function. In the response message, the most significant bit of the function code is set. The data field contains a one byte code, as described below:

code	description
01	Invalid function code
02	invalid address
03	Invalid quantity requested
04	Communication error between module and host PLC

L.8 Modbus Message Reference

Legend

SA - Slave Address (valid 1 - 247)
AH - Address (high byte of reference number)
AL - Address (low byte of reference number)
BC - Byte count (number of bytes in data portion of message)
CH - Coil number (high byte of number)
CL - Coil number (low byte of number)
DH - Data Sent (High byte)
DL - Data Sent (Low byte)
EH - Error Check (High byte of CRC)
EH - Error Check (Low byte of CRC)
Dx - Data bytes sent or received
HH - High byte of number of Holding Registers
HL - Low byte of number of Holding Registers
IH - High byte of number of inputs
IL - Low byte of number of inputs
RH - High byte of number of Input Registers
RL - Low byte of number of Input Registers
ST - Coil Status (low byte of HR0006 is read with this exception status message)
VH - Value High (FF- ON, 00- OFF)
VL - Value Low (always set to 00)
xx - Any value (00-FF)

Read Coil (Output) Status (Function Code 1)

Sent-> SA 01 AH AL CH CL EH EL
Rvd <- SA 01 BC D1 D2... Dn EH EL

D1 - First byte of coil status's returned. Low order bit is first coil, etc.

Read Input Status (Function Code 2)

Sent-> SA 02 AH AL IH IL EH EL
Rvd <- SA 02 BC D1 D2... Dn EH EL

D1 - First byte of input status returned. Low order bit is first input, etc.

Read Holding Registers (Function Code 3)

Sent-> SA 03 AH AL HH HL EH EL
Rvd <- SA 03 BC D1 D2... Dn EHEL

D1 - High byte of first register returned
D2 - Low byte of first register returned
D3 - High byte of second register returned ....
Dn - Low byte of last register returned

Read Input Registers (Function Code 4)

Sent-> SA 04 AH AL RH RL EH EL
Rvd <- SA 04 BC D1 D2... Dn EH EL

D1 - High byte of first register returned
D2 - Low byte of first register returned
D3 - High byte of second register returned . . . .
Dn - Low byte of last register returned

Write Single Coil (Function Code 5)

Sent-> SA 05 AH AL VH VL EH EL
Rvd <- SA 04 AH AL VH VL EH EL

Write Single Holding Register (Function Code 6)

Sent-> SA 06 AH AL DH DL EH EL
Rvd <- SA 06 AH AL DH DL EH EL

Read Exception Status (Function Code 7)

Sent-> SA 07 EH EL
Rvd <- SA 07 ST EH EL

Loopback Test (Function Code 8)

Sent-z- SA 08 xx xx xx xx EH EL
Rvd <- SA 06 xx x x x x xx EH EL

Preset Multiple Registers (Function Code 16 decimal)

Sent-> SA 10 AH AL HH HL D1 D2 . . . Dn EH EL
Rvd <- SA 10 AH AL HH HL EH EL

D1 - High byte of first Holding Register written to remote
D2 - Low byte of first Holding Register written to remote
Dn - Low byte of last Holding Register written to remote