OE MAX™ CONTROLS

Maximum Value for OEMs℠

NX Protocol

User Manual

Contents

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will OE Max Controls be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, OE Max Controls cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by OE Max Controls with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of OE Max Controls is prohibited.

Throughout this manual we use notes to make you aware of safety considerations.

WARNING ⚠	Identifies information about practices or circumstances which may lead to serious personal injury or death, property damage, or economic loss.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.
ATTENTION ⚠	Identifies information about practices or circumstances that can lead to minor personal injury, property damage, economic loss, or product malfunction. However, depending on situation, failure to follow the directions accompanying this symbol may also lead to serious consequences.

Safety Considerations

Please read this manual and the related documentation thoroughly and familiarize yourself with the directions before installing, operating or performing inspection and preventive maintenance. Make sure to follow the directions correctly to ensure normal operation of the product and your safety.

WARNING ⚠

- If this product is used in a situation that may cause personal injury and/or significant product damage, implement safety measures such as use of fault-safe equipment.
- Do not use this product under the conditions exposed to explosive gases. It may cause an explosion.

ATTENTION ⚠

- Make sure to use an external device when configuring the protective circuit breakers for emergencies or interlock circuits.
- Fasten the terminal screws tightly to ensure that the cable connection is secure. Incorrect cable connection may cause overheat and product malfunction.
- Operate and keep the product under the allowed conditions directed in product specifications. Otherwise it may cause overheating and product malfunction.
- Do not disassemble or remodel the product. Otherwise it may cause an electric shock or malfunction.
- Do not touch the terminals when the power is on. Otherwise it may cause an electric shock.

Installation Environment

ATTENTION ⚠

Do not install your PLC system if any of the following conditions are present.:

- Ambient temperature outside the range of 0 to 55 °C (32 to 131 °F).
- Direct sunlight.
- Humidity outside the range of 30% to 85% (non-condensing)
- Chemicals that may affect electronic parts.
- Excessive or conductive dust, or salinity.
- High voltage, strong magnetic fields, or strong electromagnetic influences.
- Direct impact and excessive vibration.

ATTENTION ⚠

Electrostatic Discharges

Under dry condition, excessive electrostatic discharges may occur. Make sure to remove electrostatic discharges by touching a grounded metal piece before touching your controller system modules.

ATTENTION ⚠

Cleaning

Never use chemicals such as thinner because they melt, deform or discolor PCB boards.

ATTENTION ⚠

Precautions for use of power

- Run your PLC system only after the I/O devices and motor devices have started. (For example, first power on in the PROG mode, then change the operation mode to RUN.)
- Make sure to power off I/O devices after ensuring PLC operation is stopped.
- If you power on/off I/O devices when the PLC system is in operation, the system may malfunction because input signal noises may be recognized as normal inputs.

ATTENTION ⚠

Before powering on

Make sure to follow these directions before powering on your PLC system.

- When installing the system, ensure that there are no metal chips or conductive fragments that stick to wiring cables.
- Ensure that power supply and I/O wirings and power supply voltage are all correct.
- Securely fasten installation and terminal screws.
- Set the operation mode switch to PROG mode.

Communication Protocols

The communication protocol provides a complete method of communications between the graphic consol programmers (WinGPC) and the PLC by controlling programs, CPU status, and I/O at user's convenience. The user can easily expand the capabilities of the overall PLC system by communicating to the PLC using a variety of peripheral communications equipment in accordance with the following communication protocols and procedures. Additionally, the communications protocol allows for the PLCs to communicate to a central computer on a single network using RS485, at a distance of up to 1.2 km (RS232C, 15 m).

- Half duplex asynchronous
- Parity: No parity
- Stop bit: 1 stop bit
- Communication method: RS232 or RS485 (optional)
- Communication speed: 4800/9600/19200/38400 bps (optional)
- Communication cable: refer to the cable wiring diagram
- Number of PLCs on a single network: Maximum of 64 (communicating 1:N using RS485)
- Maximum communication delay time: 3 seconds

Communication Flow

Step 1-Q

Query

Q (Query) is a signal sent from the peripheral devices to the PLC after setting the network ID number and the function code for the PLC to communicate with.

Step 2-QA

Query Acknowledge

QA (Query Acknowledge) is a signal sent from the PLC to the peripheral devices, indicating that the Q signal from the peripheral device was received.

Step 3-RR

Response Request

RR (Response Request) is a signal sent from the peripheral device to the PLC, indicating that the QA signal from the PLC was received. This signal is sent when Q→QA is normal.

Step 4-R

Response

When the PLC receives the RR from the peripheral device, it determines that the communication with peripherals is successful and sends R (Response) signal to the peripherals. This R signal contains how the original Q signal from the peripheral device handled its function code. The communication cycle for one function code ends when the PLC sends the R.

Communications delay

The PLC will return a signal after receiving a Q or an RR within a specific time. However, due to errors in the communications network, CRC values, and communication speed flux, there are occasions when the PLC will not receive the signal from the peripheral device. The peripheral device should allow up to three seconds for a response from the PLC. If there are no responses to the Q or the RR message, the communication is considered to have failed, and the Q or RR should be sent again.

CPU ID

All devices connected to the network need a network ID number for communication. There is an available range of 0 to 191 network ID numbers for the NX series.

Redundancy is not permitted. When a single PLC and a peripheral device are connected, usually 0, 1, or 255 is assigned as the network ID number to the PLC. When the peripheral device wants to communicate to a connected PLC regardless of its programmed network ID number, it can use global network ID number 255, to which any PLC will respond. However, the NX series can not be used to communicate with more than two CPU modules at one time, so if you assign ID 225 as an ID of more than two CPU modules at once, it will cause communication errors.

When several CPU modules are connected to one communication network, they must use individual ID numbers. The PLC's network ID number is configured using the WinGPC.

Communication steps

The NX CPU can support 2-step or 4-step communication methods. The communication methods are easily distinguished each other by selecting and sending the function code of the Q frame. Even for the 4-step method, the 2-step method can be used for the repeated function. This function sends and receives the only RR repeatedly when you want to redo the frame you sent with query, allowing users to quickly monitor data.

- **2-step communication method**
  This method allows users to easily and directly program communication since it only uses the simple
  Q → R steps.

  2-step configuration:
  Q(step 1) → R(step 2)

  Repeated function code:
  Q(step 1) → R(step 2) → RR(step 1) → R(step 2) →
  RR(step 1) → R(step 2)....

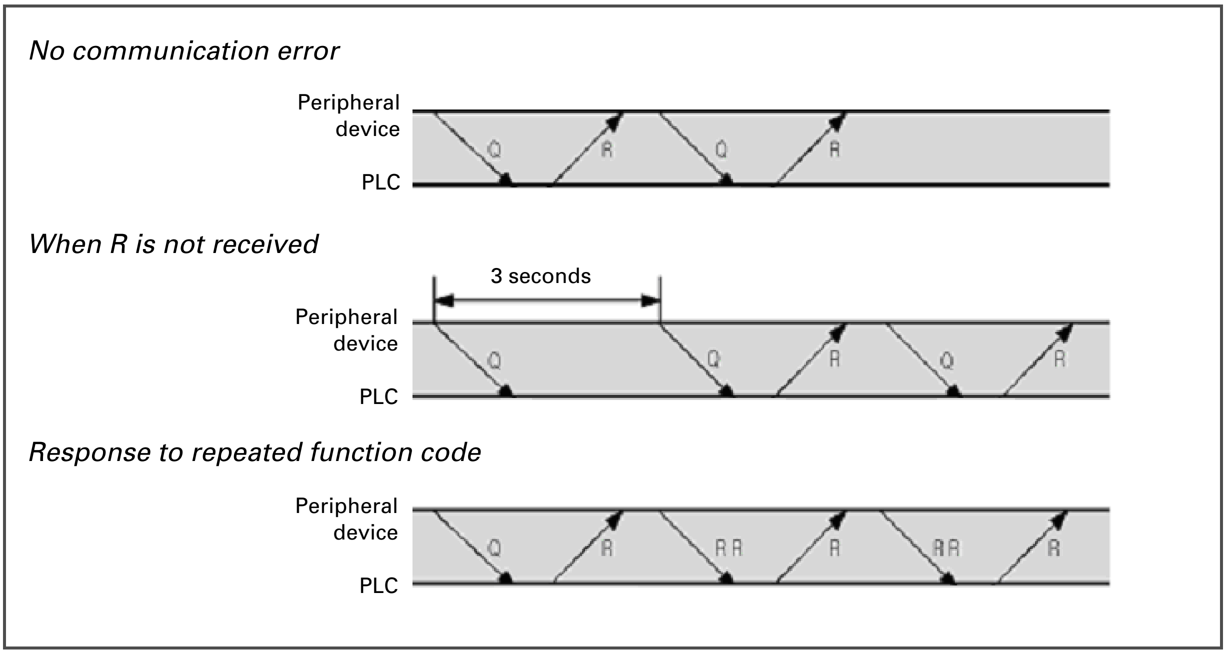
- **4-step communication method**
  Q → QA → RR → R.

  2-step method can be used for the response to the repeated function code.

  4-step configuration:
  Q(step 1) → QA(step 2) → RR(step 3) → R(step 4)

  Repeated function code:
  Q(step 1) → QA(step 2) → RR(step 3) → R(step 4) →
  RR(step 1) → R(step 2)…

2-step communication method

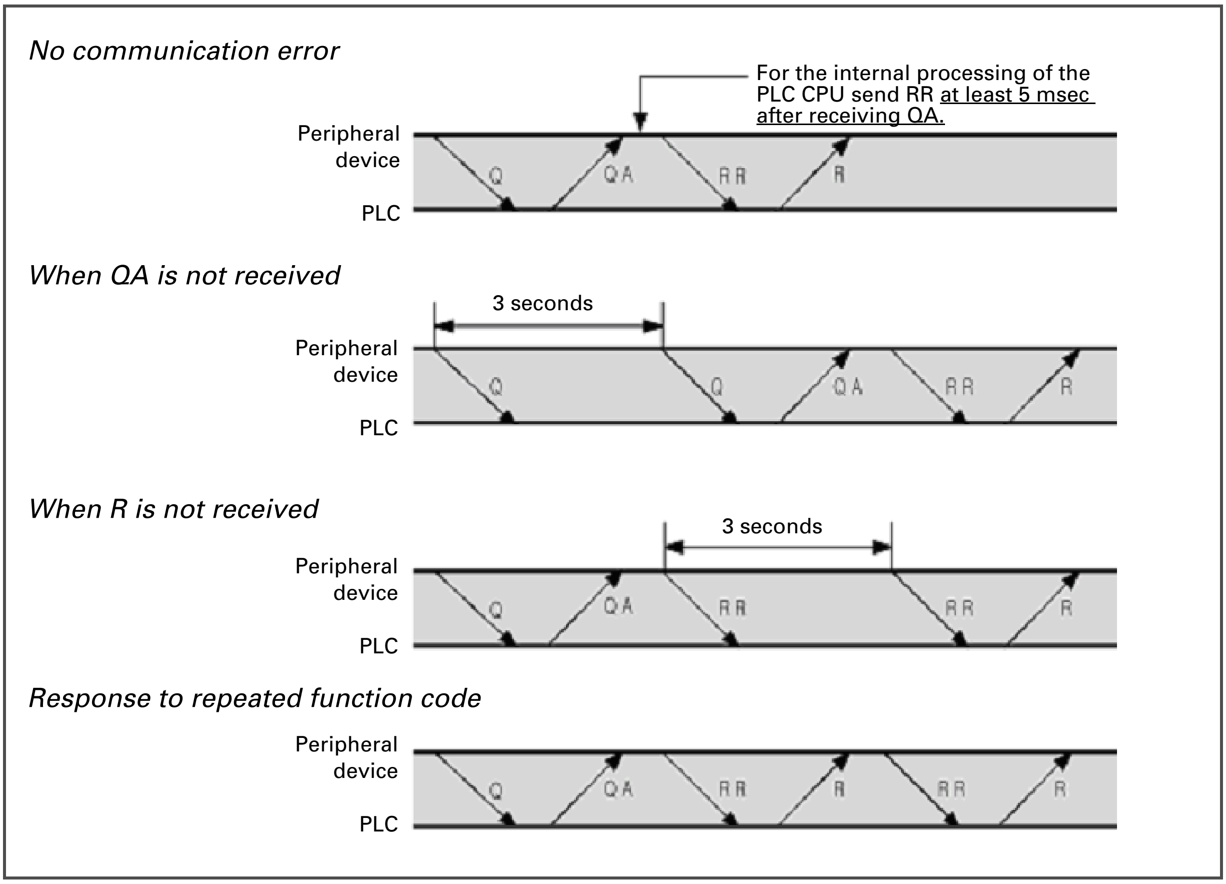


No communication error

When R is not received

Response to repeated function code

4-step communication method



No communication error

For the internal processing of the PLC CPU send RR at least 5 msec after receiving QA.

When QA is not received

When R is not received

Response to repeated function code

Function codes included in the query

Each function code is 1 byte. When the PLC receives a query (Q), the function code of the final response (R) is formed by adding $80 (hex) to the function code sent by the query. The value added to the function code sent by the query differs for 2-step and 4-step by $20 (hex).

The function code of the R message can be used by the peripheral device to verify that the correct Q message has been received by the PLC.

Communication function codes

$ notes hexadecimal notations

Communication function	Query (Q) function code		Response (R) function code		Remarks
	2-step	4-step	2-step	4-step	
Read bits	$21	$01	$A1	$81	Detailed description
Write bits	$22	$02	$A2	$82	〃
Read words	$23	$03	$A3	$83	〃
Write words	$24	$04	$A4	$84	〃
Read bits and words	$25	$05	$A5	$85	〃
Write bits and words	$26	$06	$A6	$86	〃
Read program	$27	$07	$A7	$87	No detailed description
Write program	$28	$08	$A8	$88	〃
Read instruction	$29	$09	$A9	$89	〃
Change instruction	$2A	$0A	$AA	$8A	〃
Change operand	$2B	$0B	$AB	$8B	〃
Insert instruction	$2C	$0C	$AC	$8C	〃
Delete instruction	$2D	$0D	$AD	$8D	〃
Search instruction	$2E	$0E	$AE	$8E	〃
Search operand	$2F	$0F	$AF	$8F	〃
Delete all/parts of program	$20	$10	$A0	$90	〃
No service	$00	$00	$00 (hex)	$00 (hex)	〃

The bit/word address assignment uses the absolute address method for reading memory locations.

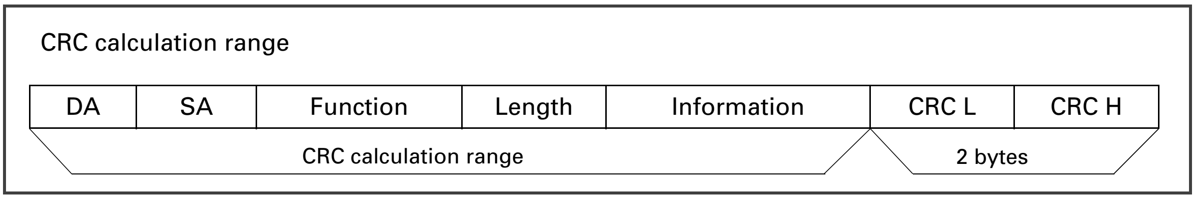
Please contact our technical support for more information about reading/writing program or other function codes.

Query, which dictionary meaning is `question’, `ask’, or `question mark’, means that a user or an application program requests any specific information to a corresponding part when it is used as a communication term.

Cyclic Redundancy Checking (CRC)

The CRC is a 2-byte checksum code attached to the end of the message by the sender to check if the communication frame is transmitted without error.

The sender calculates the CRC when it sends one-byte message, and the receiver should also calculate the CRC from the data of the message. Since this CRC calculation takes a long time when writing a communication program, you should find any ways to increase the speed of this part to avoid errors and improve the communication speed.



CRC calculation range

DA	SA	Function	Length	Information	CRC L	CRC H

CRC-16 calculation subroutine written in BASIC

```
            CRC_Sum: CRC-16 reserve code after the calculation (CRC content to be sent at the
            end of message)
            Data: CRC-16 data input to be calculated (byte data from message)

1000        CRC_Sum = CRC_Sum XOR Data
1010        FOR I=1 to 8
1020        CARRY=CRC_Sum AND 1
1030        RC_Sum=CRC_Sum SHR 1
1040        IF CARRY=1 THEN CRC_Sum XOR 0A001H
1050        NEXT I
1060        RETURN
```

CRC-16 calculation subroutine written in PASCAL

```
Procedure CRC16 (Data: Byte)
    Var i : Byte;
Begin
    CRC_Sum := CRC_Sum xor Data;
    for i : 1 to 8 do
Begin
    if((CRC_Sum and 1)=1) then CRC_Sum := (CRC_Sum shr 1) xor $A001;
                               Else CRC_Sum: = CRC_Sum shr 1;
    End;
End;
```

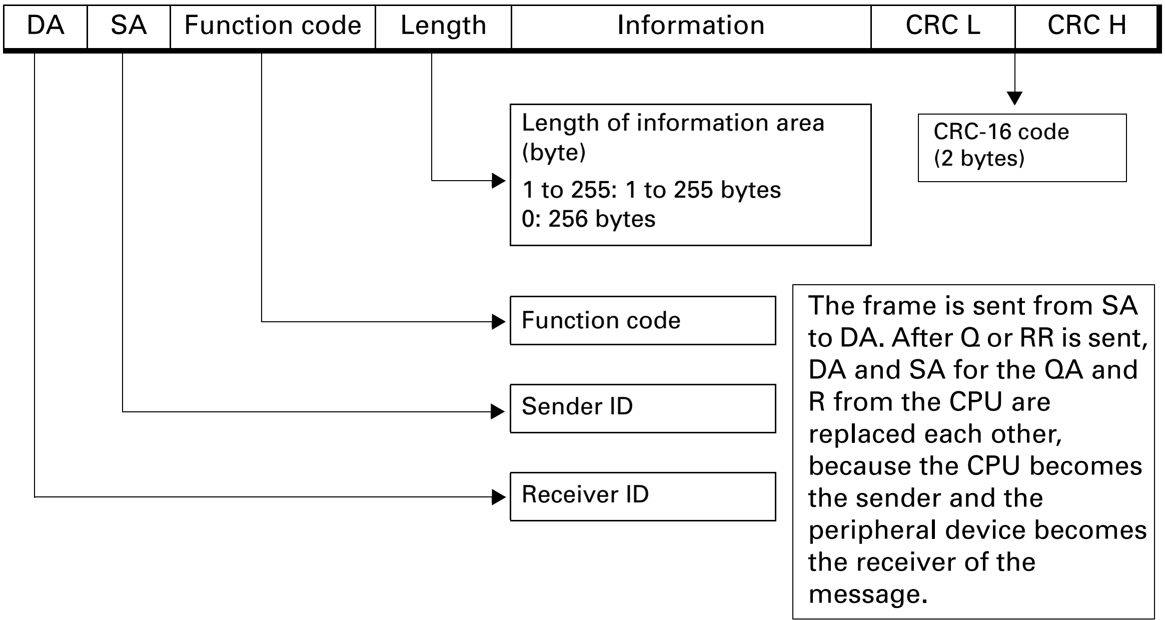
CRC-16 calculation subroutine written in C

```
Void Crc16 (unsigned int Data) {
            Unsigned int i;
            Crc=Crc^(Data & 0x00FF);
            for(i=0;i<=7;I++) {
                        if((Crc & 0x0001) == 0x0001) Crc=(Crc>>1)^0xA001;
                        else Crc=Crc>>1;
                        }
            }
```

Structure of Communication Frames

The function code is explained with the example of Query and Response frame based on the 2-step communication.

Query (Q) and Response (R) frame



DA	SA	Function code	Length	Information	CRC L	CRC H

Length of information area (byte)
1 to 255: 1 to 255 bytes
0: 256 bytes

CRC-16 code (2 bytes)

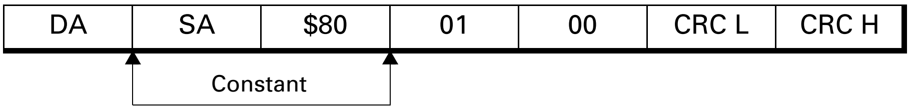
Function code

Sender ID

Receiver ID

The frame is sent from SA to DA. After Q or RR is sent, DA and SA for the QA and R from the CPU are replaced each other, because the CPU becomes the sender and the peripheral device becomes the receiver of the message.

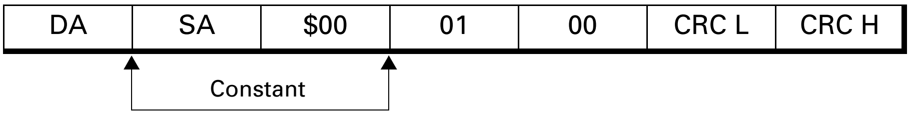
Query Acknowledge (QA) frame



DA	SA	$80	01	00	CRC L	CRC H

Constant

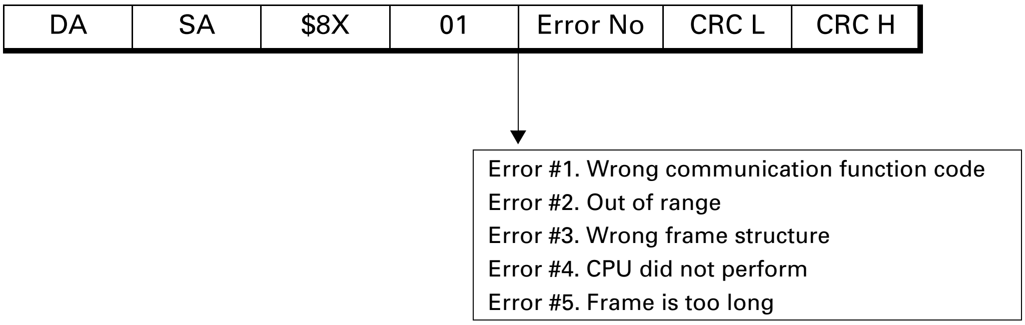
Response Request (RR) frame



DA	SA	$00	01	00	CRC L	CRC H

Constant

Response (R) frame for an error



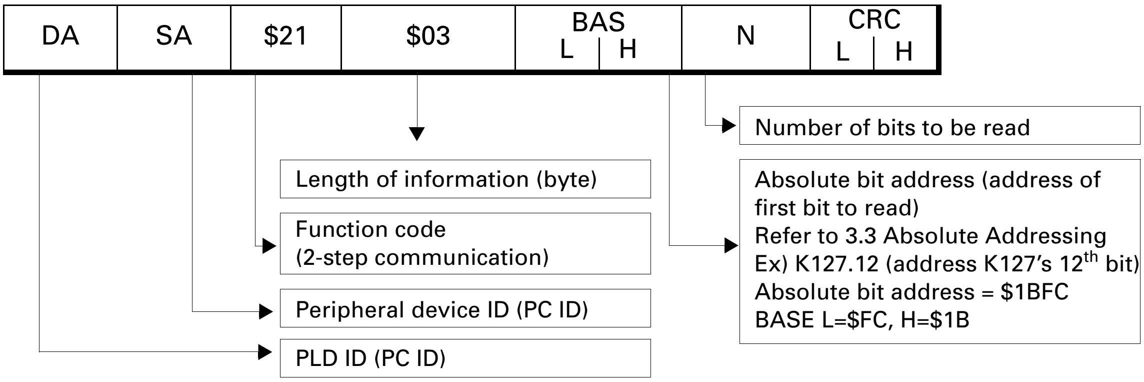
DA	SA	$8X	01	Error No	CRC L	CRC H

Error #1. Wrong communication function code
Error #2. Out of range
Error #3. Wrong frame structure
Error #4. CPU did not perform
Error #5. Frame is too long

Read bits

Read the content of the bits (R, L, M, K, F, or TC) assigned to the absolute address. Can read n consecutive bits (On/Off).

Query (Q) frame



DA	SA	$21	$03	BAS L | H	N	CRC L | H

Number of bits to be read

Absolute bit address (address of first bit to read)
Refer to 3.3 Absolute Addressing
Ex) K127.12 (address K127's 12th bit)
Absolute bit address = $1BFC
BASE L=$FC, H=$1B

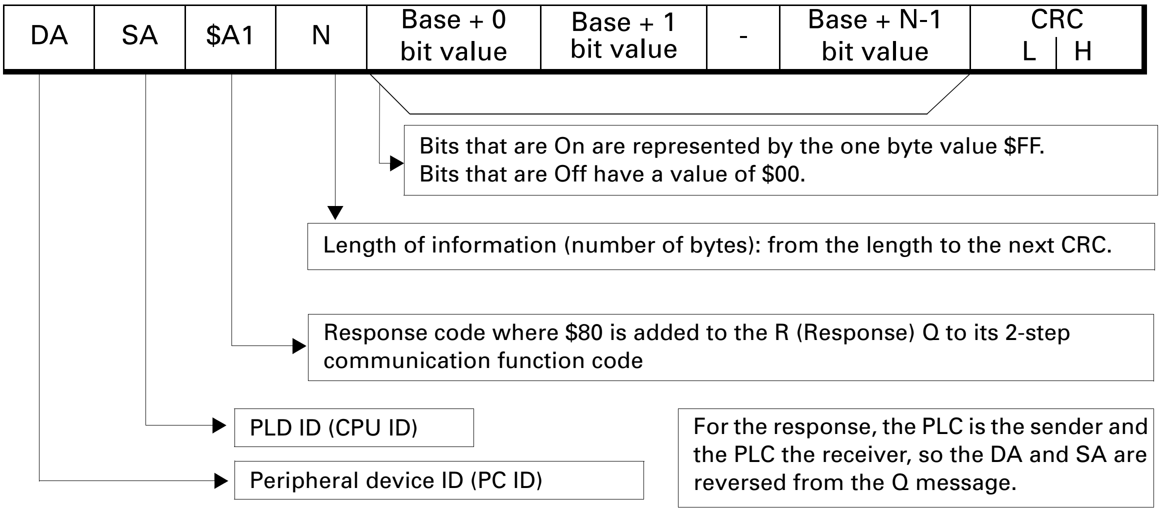
Length of information (byte)

Function code
(2-step communication)

Peripheral device ID (PC ID)

PLD ID (PC ID)

Response (R) frame



DA	SA	$A1	N	Base + 0 bit value	Base + 1 bit value	-	Base + N-1 bit value	CRC L | H

Bits that are On are represented by the one byte value $FF.
Bits that are Off have a value of $00.

Length of information (number of bytes): from the length to the next CRC.

Response code where $80 is added to the R (Response) Q to its 2-step communication function code

PLD ID (CPU ID)

Peripheral device ID (PC ID)

For the response, the PLC is the sender and the PLC the receiver, so the DA and SA are reversed from the Q message.

Write bits

Modify the contents of the bits stored in the absolute address (R, L, M, K, F, or TC).
Change the bit state between On/Off.
Can change multiple consecutive bytes.

Query (Q) frame

DA	SA	$22	N	Base		Base+0 bit value	Base+N+1 bit value	-	Base+N-3bit value	CRC	
				L	H					L	H

To turn On the desired bit value from the base, enter $FF.
To change to Off, enter $00.

Absolute bit address (starting address)

Response (R) frame

DA	SA	$A2	&01	&00	CRC	
					L	H

Fixed

Completion code

Read words

Read the content of the words (R, L, M, K, F, or W) assigned to the absolute address.
Can read n consecutive words.

Query (Q) frame

DA	SA	$23	$30	Base		N	CRC	
				L	H		L	H

Number of words to be read

Word absolute address (starting address)
Ex) Absolute address of K127 word is $01BF.
BASE L=$BF, H=$01

Response (R) frame

DA	SA	$A3	L	Base+0word		Base+1word		-	Base+1word		CRC	
				L	H	L	H		L	H	L	H

N word values from the base words requested by the Q.

Length L = Nx2

Write words

Change the content of the words (R, L, M, K, F, or W) assigned to the absolute address. Can read n consecutive words.

Query (Q) frame

DA	SA	$24	L	Base		Base+0word		-	Base+1word		CRC	
				L	H	L	H		L	H	L	H

n word values from the base words requested by the Q.

Length L = Nx2+2

Response (R) frame

DA	SA	$A2	$01	&00	CRC	
					L	H

Fixed

Read bits and words

Read the bits and/or word contents of the assigned absolute addresses.

Can read bits and words regardless of their order and location in memory.

Query (Q) frame

DA	SA	$25	L	AO		A1		-	An		CRC	
				L	H	L	H		L	H	L	H

Method of assigning bit/word absolute address

15 14 13 0

		Absolute Address(Bit/Word)

0 0 Absolute bit address
0 1 Absolute word address
1 X Not used
Ax=A0, A1, .., An Dx=D0, D1, .., Dn

Assigning absolute address for bits
Absolute address for the K127 12th bit =$1BFC
Ax=0001 1011 1111 1010
Ax L=$FC, H=$1B
Assigning addresses for word
Absolute address for the K127 word =$01BF
Ax=0100 0001 1011 1111
Ax L=$BF, H=$41

Response (R) frame

DA	SA	$A5	Lx	DO	D1		-	Dn		CRC	
					L	H		L	H	L	H

The size and location of the returned data depends on the combination of bit/word addresses requested. The Lx parameter should be checked to verify data size.

For the A0, A1, ..., An requested by the Q, the content D0, D1, ..., Dn of the word/bit is returned. If Ax denotes a bit address, the Dx data is 1 byte (On = $FF, Off = $00), and if Ax denotes a word address, the Dx data is 1 word (2 bytes).

Write bits and words

Read the bits and/or word contents of the assigned absolute addresses.
Can read bits and words regardless of their order and location in memory.

Query (Q) frame

DA	SA	$26	L	AO		DO	A1		-	CRC	
				L	H		L	H		L	H

Method of assigning bit/word absolute address

15	14	13 … 0
		Absolute Address(Bit/Word)

0 0 Absoulte bit address
0 1 Absoulte word address
1 X Not used
Ax=A0, A1, .., An Dx=D0, D1, .., Dn

Assigning absolute addresses for bits
Absolute address for the K127.12 bit = $1BFC
Ax=0001 1011 1111 1010
Assigning absolute addresses for word
The absolute address for the K127 word = $01BF
AX=0100 0001 1011 1111

When structuring the outgoing frame, be aware that the Dx of the Q changes according to the bit/word Ax type, and the L (information length) changes as well. The Dx will be either 1 or 2 bytes.

If Ax denotes a bit address, the Dx data is 1 byte (On=$FF, Off=$00), and if Ax denotes a word address, the Dx is 1 word (2 bytes).

Response (R) frame

DA	SA	$A2	$01	$00	CRC	
					L	H

Fixed

Communication Program Examples

Users can write a communication program by using the following example. For more information, contact the sales or technical department.

<table>
<tr><th>Program</th><th>Notes</th></tr>
<tr><td><b><PLC communication sample code></b>
#include <stdio.h>
#include <dos.h>
#include <conio.h></td><td>This program was written in Borland C++. It uses the peripheral devices such as PC to read M000 to M127 words, stores them in the K000 to K127, and then compares the two registry values and indicates the results on the screen using the OK or the FAIL notation. The user may read or manipulate the various communication function codes and the sent/received information to control the PLC in various ways.</td></tr>
<tr><td>#define PC_ID 0xE2
#define time_limit 28
#define retrial_limit 2
#define TRUE 1
#define FALSE 0
#define lower_byte(x) (unsigned int) ((x)& 0x00FF)
#define upper_byte (x) (unsigned int) (((x)& 0xFF00)>>8)</td><td>This program consists of a header, the main program, and various functions. The buffers and variables needed to store the communication data are set as global variables, so that the main and various other functions may reference them.
By using the COM1 and COM2 ports of the computer, serial communication is possible. By using the GPU-300 card, parallel communication is also enabled.</td></tr>
<tr><td>typedef int BOOL;
unsigned int PORTADD,DIVISOR,sending_delay, receiving_delay;
unsigned int sending_frame[262],receiving_frame[262];
unsigned int Crc;</td><td>The Qs, QAs, RRs, and Rs are handled in the job function. If there is any communication delay or frame breakdown, retry 3 times, then issue a communication error.</td></tr>
<tr><td>unsigned int card,i,ix,iy,smode;
unsigned int port_number;
unsigned int PlcID,OldID;
BOOL Success;
unsigned int data,JobID,retrialC;
unsigned int Old,New,receiving_Index_max,sending_Index_max, index,watchdog;
unsigned int M[128],K[128]; /* Example Register */</td><td>The procedure of the communication, according to the JobID is:
1.Q sending
2.QA receiving
3.RR sending
4.R receiving
When an error occurs in a frame, a retransmission should be made.</td></tr>
<tr><td>void RR_occurring(void);
void Trsport(unsigned int);
unsigned int Recport(void);
BOOL sending_occuring(void);
BOOL receiving_occuring (void);
void Crc16(unsigned int);
void Job(void);
unsigned int communication(void);
void Mword_reading(void);
void Kword_writing(void);</td><td><b><Main operations of the program></b>
1. Adjusts the initial communication port and the board rate for communication. Then initializes the variables.
2. Using the communication function codes, reads the data of the M field, reads the word values of the M0 to M127 word area. The K registers are the retentive registers.
3. Uses the communication code to read the data of the K area.
4. Compares the values of the M area and the values of the K area, and indicates OK when the values are the same.</td></tr>
<tr><td>void main(void)
{
unsigned int i;
/* Selection of communication port */
clrscr();
printf("PORT : COM1[1]/ COM[2]/ GPC-232[3]/GPC-485[4]/GPC-Parallel[5] = ");
scanf("%d",&port_number);
if ((port_number < 1) || (port_number > 5)) port_number=5;</td><td><b>Beginning of the main program</b>
Select the port of the peripheral device for the communication:
Serial 9 pins, 25 pins
Parallel GPU-300 parallel port</td></tr>
<tr><td>/* Selection of Baudrate for Serial communication */
sending_delay=10;
if (port_number != 5)
{
printf("GPC card BAUD-RATE : 9600[1]/ 4800[2]/ 2400[3] = ");
scanf("%d",&i);
if ((i < 1) || (i > 3)) i=1;
if (i == 3) i=4;
if ((port_number == 1) || (port number == 2)) DIVISOR=12 * i;
else DIVISOR=40 * i;
receiving_delay=3 * i + 1;
}</td><td>Select board rate:
9600 bps (max):
4800 bps
2400 bps
Set the communication environment (delay time) for the selected ports.
Note: GPC-300 card port address = 0x0300</td></tr>
</table>

Program	Notes

* Initialization of GPC card */
if(port_number == 1) PORTADD=0x3F0;
if(port_number == 2) PORTADD=0x2F0;
if ((port_number >= 3) && (port_number <=5))
{
PORTADD=0x300;
outportb(0x303,0xC0);/* Mode=2 of 8255 */
outportb(0x303,0x05);/* PC2=1 of 8255 :Disable IRQ2 */
outportb(0x301,0xFF);/* PB0=1 of 8255 :Sending Enable RS-485*/
outportb(0x303,0x01);/* PC0=1 of 8255 :Serial Input Enable*/
if(port_number == 3) outportb(0x303,0x02);/* PC1=0 of 8255 :Select RS-232 */
if(port_number == 4) outportb(0x303,0x03);/* PC1=1 of 8255 :Select RS-485 */
if(port_number == 5) outportb(0x303,0x00);/* PC0=0 of 8255 :Disable SerialInput*/
}
else
outportb(PORTADD+0x09,(inportb(PORTADD+0x09)&0xF0));/ *Disable Interrupt*/
/* Initialization of USART-Chip : 8250 */
if (port_number != 5)
{
outportb(PORTADD+0x0B,0x80);/* Set of DLAB=1 */
outportb(PORTADD+0x09,0x00);/* Set of High Byte DIVISOR */
outportb(PORTADD+0x08,DIVISOR);/* Set of Low Byte DIVISOR */
outportb(PORTADD+0x0B,0x03); /* Parity=None/Stop=1/ Length=8 */
}
/* Processing communication of Read & Write */
for(; ;)
{
	GPC-300 card Setting (8255chip setting)
	Uses the communication card that is connected, and sets the environment according to the PLC communication specifications, so that communication is possible.

printf("----------------\nPLC-ID (CPU ID) :");
scanf("%d",&PlcID);
	CPU-ID: Input PLC ID (0 to 255)

Mword_reading();
Kword_writing();
}
}
	Reads the register value for the M area (M0 to M127)
	Stores the value for the M area in the K area. (K0 to K127)

void RR_occuring(void)
{
receiving_frame[2]=0;
receiving_frame[3]=1;
receiving_frame[4]=0;
}
	RR (Request Response) request function

void Trsport(unsigned int data)
{
if (port_number == 5) outportb(PORTADD,data);
else outportb(PORTADD+0x08,data);
}
	Sends data to the communication port.

unsigned int Recport(void)
{
unsigned int dt;
if (port_number == 5) dt=inportb(PORTADD);
else dt=inportb(PORTADD+0x08);
return(dt);
}
	Reads the received data from the communication port.

BOOL sending_occuring(void)
{
BOOL tf;
if (port_number == 5) tf=((inportb(PORTADD+0x02) & 0x80)==0x80);
else tf=((inportb(PORTADD+0x0D) & 0x20)==0x20);
return(tf);
}
	Outputs the data when a Send event occurs..

BOOL receiving_occuring(void)
{
BOOL rf;
if (port_number == 5) rf=((inportb(PORTADD+0x02) & 0x20)==0x20);
else rf=((inportb(PORTADD+0x0D) & 0x01)==0x01);
return(rf);
}
	Inputs the data when a Receiver event occurs.

void Crc16(unsigned int data)
{
unsigned int i;
Crc=Crc^(data & 0x00FF);
for(i=0;i<=7;i++)
{
if((Crc & 0x0001) == 0x0001) Crc=(Crc>>1)^0xA001; /* 0x0001 : multi-nominal expression */
else Crc=Crc>>1;
}
}
	CRC calculation
	Encodes the communication data in the byte stream. Any completed communication function will be attached to the latest frame or will be compared with the attached CRC to check for data errors.
	(Note: The CRC method can be implemented in several ways within the rule specified as shown in the left code.)

Program	Notes

void Job(void)
{
/* JobID=0 : Change to sending-mode for serial port */
/* JobID=1 : Transmit sending-frame */
/* JobID=2 : Change to receiving-mode for serial port */
/* JobID=3 : Address polling of ACK from CPU */
/* JobID=4 : Receive ACK from CPU */
/* JobID=5 : Change to sending-mode for serial port */
/* JobID=6 : Transmit RR-Frame */
/* JobID=7 : Change to receiving-mode for serial port */
/* JobID=8 : Address polling of RES from CPU */
/* JobID=9 : Receive RES from CPU */
/* JobID=10 : Success communication processing */
switch(JobID)
{
	Communication sequence function
	JobID=0 to 4: handle Q and QA frames
	JobID=5 to 9: handle RA and R frames

case 0: case 5:if (port_number != 5)
{
if (port_number == 4) outportb(0x301,0xFF);
else outportb(PORTADD+0x0C,(inportb(PORTADD+0x0C) | 0x02));
delay(sending_delay);
}
if (JobID == 5) RR_occuring();
watchdog=0; index=0; sending_Index_max=5; Crc=0xFFFF;
JobID++;
break;
	JobID 0,5:
	A frame that sends the data from the peripheral device to the PLC. It resets the watchdog and the CRC. Use a delay after the send to avoid errors due to communications delays.

case 1: case 6:if (receiving_occuring()) data=Recport();
if (sending_occuring())
{
if (index<sending_Index_max-1)
{
Trsport(receiving_frame[index]);
Crc16(receiving_frame[index]);
if (index==3)
{
if (receiving_frame[3]==0) sending_Index_max=256+5;
else sending_Index_max=receiving_frame[3]+5;
}
}
else if (index==sending_Index_max-1)
{
receiving_frame[index]=lower_byte(Crc);
Trsport(receiving_frame[index]);
}
else if (index==sending_Index_max)
{
receiving_frame[index]=lower_byte(Crc);
Trsport(receiving_frame[index]);
}
else if (index==sending_Index_max)
{
receiving_frame[index]=upper_byte(Crc);
Trsport(receiving_frame[index]); watchdog=0; JobID++;
}; index++;
}
break;
	JobID 1,6:
	Sends the Q and RR data. When there is no error, it resets the watchdog and proceeds on to the next sequence.

case 2: case 7:if (port_number != 5)
{
delay(receiving_delay);
if (port_number ==4) outportb(0x301,0x00);
else outportb(PORTADD+0x0C,(inportb(PORTADD+0x0C) & 0xFD));
}
JobID++;
break;
	JobID=2,7:
	A sequence that senses the sending of the QA and R data to the peripheral device after the completion of the functions that are received by the PLC from the previous frame.

case 3:
case 8:if (receiving_occuring())
{
data=Recport();
if(data==PC_ID)
{
Crc=0xFFFF; index=1; receivingIndexmax=5;
receiving_frame[0]=data; Crc16(data); JobID++;
}
}
	JobID=3,8:
	Handles the received data, and calculates the CRC of the received data.

Program	Notes

break;
case 4:
case 9:if(receiving_occuring())
{
if(index<receiving_Index_max-1)
{
receiving_frame[index]=Recport();
Crc16(receiving_frame[index]);
if(index==3)
{
if(receiving_frame[3]==0) receiving_Index_max=256+5;
else receiving_Index_max=receiving_frame[3]+5;
}
}
else if(index==receiving_Index_max-1)
{
receiving_frame[index]=Recport();
if(receiving_frame[index]!=lower_byte(Crc)) JobID=(JobID & 0x05);
}
else if(index==receiving_Index_max)
{
receiving_frame[index]=Recport();
if(receiving_frame[index]==upper_byte(Crc)) JobID++;
else JobID=(JobID & 0x05);
}; index++;
}
break;
	JobID=4,9:
	Stores the received data in the internal receive buffer and compares the CRC value sent by the PLC to the calculated CRC value. It notifies the system that a successful communication is made when the two values match, and proceeds on to the next sequence.

case 10:Success=TRUE;
}
}
	JobID=10:
	Notifies the successful sending and receiving

unsigned int communication(void)
{
struct time t;
unsigned far *tm;
int ret;
Success=FALSE;
receiving_frame[0]=PlcID; receiving_frame[1]=PC_ID;
retrialC=retrial_limit;
watchdog=0; JobID=0; index=0; sending_Index_max=5; Crc=0xFFFF;
do
{
tm=(unsigned far *) 0x046C;
New=*tm;
Job();
if(watchdog>Time_limit)
{
watchdog=0; retrialC--;
JobID=(JobID & 0x05);
}
if(!(((Old^New) & 0x02)==0))
{
watchdog=watchdog+1;
Old=New;
}
}while((retrialC!=0) && (Success==FALSE));
if(retrialC==0) ret=1;
else ret=0;
return(ret);
}
	If the frames that were sent have no response within 3 seconds, assumes it failed communication, and retransfers the data.
	The time from the sending and receiving is counted using the watchdog timer.
	Resets the watchdog time when a retransfer is being made. No response after 3 transmissions indicates a communication error. (Normal return value = 0, Abnormal return value = 1)

void Mword_reading(void)
{
/* Example of Read-Register */
int i;
receiving_frame[2]=3;/* EXAMPLE READ WORD(M000-M0127) */
receiving_frame[3]=3;/* Number Of Byte For Information = 3 */
receiving_frame[4]=0xC0;/* BASE(M000=$00c0) */
receiving_frame[5]=0;/* BASE HIGH */
receiving_frame[6]=128;/* Number Of Byte M000-M127 */
if(communication() == 0)
{
printf("READ M0000-M0127 OK\n");
for(i=0;i<=127;i++) M[i]= receiving_frame[i*2+4] + receiving_frame[i*2+5]*256;
}
else printf("communication error\n");
}
	Reading function of the M register
	Uses the communication function code 3 (reading N consecutive words) to read the M area.
	Note:
	Sending frame [4] = The lower byte of the absolute address of the words to be read.
	Sending frame [5] = The upper byte of the absolute address of the word to be read.
	Absolute address of M0 = 0x0C0
	Note: Sending frame [6] = The number of words to be read.
	Sends a function code requesting to read the M area, and stores the received data in the buffer.

void Kword_writing(void)
{
/* Example of Write-Register */
int i;
receiving_frame[2]=4;   /* EXAMPLE write WORD(K000-K063) */
receiving_frame[3]=130;   /* Number Of Byte For Information */
receiving_frame[4]=0x40;   /* BASE(K000=$0140) LOW */
receiving_frame[5]=1;   /* BASE HIGH */
for(i=0;i<=63;i++)
{
receiving_frame[i*2 +6]= lower_byte(K[i]);
receiving_frame[i*2 +7]= upper_byte(K[i]);
}
	Writing Function of the K Register
	Uses the communication function code 4 (writing N consecutive words) to store the specified value in the K000 to K063 word.
	Note:
	Absolute address of K0 = 0x0140

Program	Notes

if(communication() == 0) printf("WRITE K0000-K0063 OK\n");
else printf("communication error\n");
receiving_frame[2]=4;   /* EXAMPLE write WORD(K064-K0127) */
receiving_frame[3]=130;   /* Number Of Byte For Information */
receiving_frame[4]=0x80;   /* BASE(K000=$0180) LOW */
receiving_frame[5]=1;   /* BASE HIGH */
for(i=0;i<=63;i++)
{
receiving_frame[i*2 +6]= lower_byte(K[i+64]);
receiving_frame[i*2 +7]= upper_byte(K[i+64]);
}
if(communication() == 0) printf("WRITE K0064-K0127 OK\n");
else printf("communication error\n");
}
	Writing function of the K Register
	Uses the communication function code 4 (writing N consecutive words) to store the specified value in the K064 to K127 word.
	Note:
	Absolute address of K64 = 0x0180

NX Protocol

OE MAX Controls

www.oemax.com

Publication OEMAX-NX70-UM012A-EN-E-July 2005