ZL-69X

Modbus Protocol V3.5

Interface

Serial port: RS485
Frame transmission: RTU
Frame format: 1 start bit, 8 data bits, no parity bit, 1 stop bit, CRC checksum.
Baud rate: 2400, 4800, 9600 and 19200 optional.

Function Code

Function codes

Function Code	Description	Definition
0x01	Read Coils	This function code reads status from 1 to 2000 contiguous coils in a remote device.
0x02	Read Discrete Inputs	This function code reads status from 1 to 2000 contiguous discrete inputs in a remote device.
0x03	Read Holding Registers	This function code reads the contents of a contiguous block of holding registers in a remote device.
0x04	Read Input Registers	This function code reads from 1 to approx.125 contiguous input registers in a remote device.
0x05	Write Single Coil	This function code writes a single output to either ON or OFF in a remote device.
0x06	Write Single Register	This function code writes a single holding register in a remote device.
0x10	Write Multiple Registers	This function code writes a block of contiguous register (1 to approx. 120 registers) in a remote device.

Error codes

Error Code	Description	Definition
0x01	Invalid function code	The received function code is not among the effective ones
0x02	Invalid data address	The received data address is not within defined ones
0x03	Invalid data value addressing	The data address invalid
0x04	Write failure.	Fail to write to register
0x05	Too busy now to receive data	Can not execute the required function now
0x06	Frame too long	The frame exceeds 255 bytes
0x0C	CRC error	

Effective function codes for this controller

Code	Description
0x01	Read coil
0x03	Read multiple registers
0x10	Write multiple registers
0x2B	Read device ID

Message

READ COIL (0x01)

Function code 0x01 request：

Definition	Address	Function code	Start register	Register qty.	CRC checksum
Data	ADDR	01H	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Function code 0x01 response:

Definition	Address	Function code	Response data bytes	Response data	CRC checksum
Data	ADDR	01H	X	DATA	CRC16
Bytes	1	1	1	X	2

Example:

Request		Response	
Field Name	(Hex)	Field Name	(Hex)
Function	1	Function	1
Starting Address Hi	0	Byte Count	1
Starting Address Lo	13	Outputs status 27-20	CD
Quantity of Outputs Hi	0	Outputs status 35-28	6B
Quantity of Outputs Lo	13	Outputs status 38-36	5

READ DISCRETE INPUTS (0x02)

Function code 0x02 request:

Definition	Address	Function code	Start register	Register qty.	CRC checksum
Data	ADDR	02H	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Function code 0x02 response:

Definition	Address	Function code	Response data bytes	Response data	CRC checksum
Data	ADDR	02H	X	DATA	CRC16
Bytes	1	1	1	X	2

Example:

Request		Response	
Field Name	(Hex)	Field Name	(Hex)
Function	2	Function	2
Starting Address Hi	0	Byte Count	3
Starting Address Lo	C4	Inputs status 204-197	AC
Quantity of Inputs Hi	0	Inputs status 212-205	DB
Quantity of Inputs Lo	16	Inputs status 218-213	35

READ HOLDING REGISTERS (0x03)

Function code 0x03 request:

Definition	Address	Function code	Start register	Register qty.	CRC checksum
Data	ADDR	03H	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Function code 0x03 response:

Definition	Address	Function code	Response data bytes	Response data	CRC checksum
Data	ADDR	03H	X	DATA	CRC16
Bytes	1	1	1	X	2

Example:

Request		Response	
Field Name	(Hex)	Field Name	(Hex)
Function	3	Function	3
Starting Address Hi	0	Byte Count	6
Starting Address Lo	6B	Register value Hi (108)	2
No.of Registers Hi	0	Register value Lo (108)	2B
No.of Registers Lo	3	Register value Hi (109)	0
		Register value Lo (109)	0
		Register value Hi (110)	0
		Register value Lo (110)	64

READ INPUT REGISTERS (0x04)

Function code 0x04 request:

Definition	Address	Function code	Start register	Register qty.	CRC checksum
Data	ADDR	04H	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Function code 0x04 response:

Definition	Address	Function code	Response data bytes	Response data	CRC checksum
Data	ADDR	04H	X	DATA	CRC16
Bytes	1	1	1	X	2

Example:

Request		Response	
Field Name	(Hex)	Field Name	(Hex)
Function	4	Function	4
Starting Address Hi	0	Byte Count	2
Starting Address Lo	8	Input Reg.9 Hi	0
Quantity of Input Reg.Hi	0	Input Reg.9 Lo	0A
Quantity of Input Rge.Lo	1		

WRITE SINGLE COIL (0x05)

Function code 0x05 request:

Definition	Address	Function code	Start register	Register value	CRC checksum
Data	ADDR	05H	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Function code 0x05 response:

Definition	Address	Function code	Start register	Register value	CRC checksum
Data	ADDR	05H	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Example:

Request		Response	
Field Name	(Hex)	Field Name	(Hex)
Function	5	Function	5
Output Address Hi	0	Output Address Hi	0
Output Address Lo	AC	Output Address Lo	AC
Output Value Hi	FF	Output Value Hi	FF
Output Value Lo	0	Output Value Lo	0

WRITE SINGLE REGISTER (0x06)

Function code 0x06 request:

Definition	Address	Function code	Start register	Register value	CRC checksum
Data	ADDR	6	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Function code 0x06 response:

Definition	Address	Function code	Start register	Register value	CRC checksum
Data	ADDR	6	Start REG	N REG	CRC16
Bytes	1	1	2	2	2

Example:

Request		Response	
Field Name	(Hex)	Field Name	(Hex)
Function	6	Function	6
Register Address Hi	0	Register Address Hi	0
Register Address Lo	1	Register Address Lo	1
Register Value Hi	0	Register Value Hi	0
Register Value Lo	3	Register Value Lo	3

Write Multiple registers (0x10)

Function code 0x10 request:

Definition	Address	Function code	Start register	Register qty.	Byte qty.	Register value	CRC checksum
Data	ADDR	16	Start REG	Quantity	Byte Cnt	N REG	CRC16
Bytes	1	1	2	2	1	X	2

Function code 0x10 response:

Definition	Address	Function code	Start register	Register qty.	CRC checksum
Data	ADDR	16	Start REG	Quantity	CRC16
Bytes	1	1	2	2	2

Example:

Request		Response	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	10	Function	10
Starting Address Hi	0	Starting Address Hi	0
Starting Address Lo	1	Starting Address Lo	1
Quantity of Registers Hi	0	Quantity of Registers Hi	0
Quantity of Registers Lo	2	Quantity of Registers Lo	2
Byte Count	4		
Registers Value Hi	0		
Registers Value Lo	0A		
Registers Value Hi	1		
Registers Value Lo	2		

Read Device Identification(0x2B)

Request PDU:

Function code	1 Byte	**0x2B**
MEI Type	1 Byte	0x0E
Read Device ID code	1 Byte	01 / 02 / 03 / 04
Object Id	1 Byte	0x00 to 0xFF

Response PDU:

Function code	1 Byte	**0x2B**
MEI Type	1 byte	0x0E
Read Device ID code	1 Byte	01 / 02 / 03 / 04
Conformity level	1 Byte	
More Folows	1 Byte	00 / FF
Next Object Id	1 Byte	Object ID number
Number of objects	1 Byte	
List Of		
Object ID	1 Byte	
Object length	1 Byte	
Object Value	1 Byte	

Example:

Request		Response	
Field Name	*Value*	*Field Name*	*Value*
Function	2B	Function	2B
MEI Type	0E	MEI Type	0E
Read Dev Id code	01	Read Dev Id Code	01
Object Id	00	Conformity Level	01
		More Follows	00
		NextObjectId	00
		Number Of Objects	03
		Object Id	00
		Object Length	16
		Object Value	"Companyidentification"
		Object Id	01
		Object Length	0A
		Object Value	" Product code "
		Object Id	02
		Object Length	05
		Object Value	"V2.11"

Remark:

CRC16 checksum is for: address, function code, start register/response data bytes, register qty./response data.

CRC16 checksum sending sequence: low byte 1st.

COIL DATA POINTS LIST

Function code: 0x01

Address	Bit	Description	R/W	Remark
1	Bit	Room temperature sensor failure	R	1: Alarm, 0: Stop
2	Bit	Evaporator sensor failure	R	1: Alarm, 0: Stop
3 ~ 6	Bit	Reserved		
7	Bit	EEPROM setting data error	R	1: Alarm, 0: Stop
8	Bit	Reserved		
9	Bit	High temperature alarm	R	1: Alarm, 0: Stop
10	Bit	Low temperature alarm	R	1: Alarm, 0: Stop
11/12	Bit	Reserved		
13	Bit	Pressure alarm	R	1: Alarm, 0: Stop
14	Bit	Lack of freon protection	R	1: Alarm, 0: Stop
15	Bit	Evaporator frozen protect	R	1: Alarm, 0: Stop
16	Bit	Reserved		
17	Bit	Compressor status	R	1: ON, 0: OFF
18	Bit	Reserved		
19	Bit	Heater status	R	1: ON, 0: OFF
20	Bit	Evaporator fan status	R	1: ON, 0: OFF
21 ~ 32	Bit	Reserved		

6、REGISTER DATA POINTS LIST

Function code 0x06, 0x10: Write
Function code 0x03: Read

Address	Type	Description	R/W	Range	Remark
1	int	Room temperature	R	-40 ~ 130℃	Temperature * 10
2	int	Evaporator temperature	R	-40 ~ 130℃	Temperature * 10
3 ~ 8	int	Reserved			
9	int	Setpoint for cooling	R/W	15 ~ 50℃	Temperature * 10
10	int	Setpoint for heating	R/W		Temperature * 10
11	int	Hysteresis	R/W	1 ~ 20℃	Temperature * 10
12	int	High temperature alarm point	R/W	-5 ~ 60℃	Temperature * 10
13	int	Low temperature alarm point	R/W		Temperature * 10
14	int	Evaporator fan setpoint	R/W	0 ~ 40℃	Temperature * 10
15	int	Evaporator fan hysteresis	R/W	1 ~ 20℃	Temperature * 10
16	int	Evaporator protection point	R/W	-5.0 ~ 20.0℃	Temperature * 10
17 ~ 18	int	Reserved			
19	int	Room sensor calibration	R/W	-3.0 ~ 3.0℃	Temperature * 10
20	int	Evaporator sensor calibration	R/W	-3.0 ~ 3.0℃	Temperature * 10
21	int	High temperature alarm	R/W	0/1	0: disable, 1: enable
22	int	Low temperature alarm	R/W	0/1	0: disable, 1: enable
23	int	Lack of freon alarm	R/W	0/1	0: disable, 1: enable
24 ~ 29	int	Reserved			
30	int	Evaporator sensor works	R/W	0/1	0: disable, 1: enable
31	int	Compressor works	R/W	0/1	0: enable compressor, 1: disable compressor
32	int	Heater works	R/W	0/1	0: enable heater, 1: disable heater
33	int	Evaporator fan working mode	R/W	0/1	0: temperature control, 1: keep energized.
34	int	Reserved			
35	int	Digital input DI mode	R/W	0/1	0: NC, normal close, 1: NO, normal open
36	int	Reserved			
37	int	Buzzer works	R/W	0/1	0: disable, 1: enable
38	int	Compressor delay protection time	R/W	0 ~ 5 min	
39	int	Compressor minimum run time	R/W	0 ~ 5 min	
40 ~ 41					
42	int	On/off	R/W	0xFF00/0x0000	0x0000: off, 0Xff00: on. For ZL-690/691 only
43 ~ 48	int	Reserved			

Remark:

Temperature record example:

27.5℃ = 27.5 x 10 = 275 = 0x0113

-9.9℃ = -9.9 x 10 = -99 = 0xFF9D

The address in the table is PLC address. For modbus protocol address, -1.

Coils can be read continuously for 32 units.

Registers can be read continuously for 48 units, and written continuously for 40 units.

7、CRC16 Checksum

In message, CRC checksum's low byte 1st

```
unsigned int CRC_Check(unsigned char *puchMsg, unsigned int usDataLen)
{
   unsigned char uchCRCHi = 0xFF; // high byte of CRC initialized
   unsigned char uchCRCLo = 0xFF; // low byte of CRC initialized
   unsigned char uIndex;                  // will index into CRC lookup table

   while (usDataLen--)                    // pass through message buffer
   {
         uIndex = uchCRCLo ^ *puchMsg++; // calculate the CRC
         uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex];
         uchCRCHi = auchCRCLo[uIndex];
   }
```

```
    return (uchCRCHi << 8 | uchCRCLo);
}


// Table of CRC values for high order byte:
const unsigned char auchCRCHi[] =
{
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
};

// Table of CRC values for low order byte:
const unsigned char auchCRCLo[] =
{
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
};
```