1 Communication protocols

1.1 Modbus protocol

Modbus is a serial communication protocol became a de facto standard in industrial communication, and is now the most widely used connection protocol among industrial electronics devices. It is a protocol based on request/response and offers services specified by function codes.

SlimLine supports the Modbus RTU protocol on the serial ports and Modbus over IP with Ethernet connection on port ***502***. The Modbus RTU protocol on the serial port has the default communication parameters ***115200, e, 8, 1*** and the node address for both serial port and TCP/IP is ***1***.

1.1.1 Access to variables from modbus

The modbus functions allow to access to ***MX100*** user memory. The supported functions are:

Code	Function	Object type	Access type	Address range
01h	Read coil status	Single bit	Read	40000-44095 (20000-24095) ***(Note 1)***
02h	Read input status	Single bit	Read	40000-44095 (20000-24095) ***(Note 1)***
03h	Read holding registers	Word (16 Bit)	Read	40000-42047 (20000-22047) ***(Note 2)***
04h	Read input registers	Word (16 Bit)	Read	40000-42047 (20000-22047) ***(Note 2)***
05h	Force single coil	Single bit	Write	40000-44095 (20000-24095) ***(Note 1)***
06h	Preset single register	Word (16 Bit)	Write	40000-42047 (20000-22047) ***(Note 2)***
10h	Preset multiple registers	Word (16 Bit)	Write	40000-42047 (20000-22047) ***(Note 2)***

From software version SFW167D000 the addressable area is also in the range from 20000 to 2xxxx.

Note 1) In the functions that access the single-bit (every bit equals to one byte of memory), the address of the variable in the command is used. So having to access to the ***MX100.50*** location, the address value will be ***40050***.

Note 2) In the functions that access the registers (16 bits) the address of the variable divided by 2 is used. So having to reach the ***MX100.50*** location, the value ***40025*** will be used.

1.1.2 Reading variables from modbus

To read variables, the ***Read Holding registers*** (code ***0x03***) command is used. Assuming you have to read a ***DWORD*** variable allocated in memory at ***MX100.64*** address, this is the formula to calculate the address:

((Address of variable/2)+Offset)-1 → ((64/2)+40000)-1=40031 → 0x9C5F

Being a ***DWORD*** variable, we read 2 consecutive registers starting from address allocation. Assuming that the value of the variable is ***0x12345678***, we have:

Modbus RTU frames

Command frame: ***01 03 9C 5F 00 02 DA 49***
Answer frame: ***01 03 04 56 78 12 34 66 D5***

Modbus TCP/IP frames

Command frame: ***00 00 00 00 00 06 01 03 9C 5F 00 02***
Answer frame: ***00 00 00 00 00 07 01 03 04 56 78 12 34***

The representation of data in SlimLine is ***Little-Endian***. The numbering starts from the least significant byte and ending with the most significant. So as you can see from the response string, the value of the 32-bit variable ***0x12345678*** is returned in two 16-bits registers with values ***0x5678***, ***0x1234***.

1.1.3 Writing variables from modbus

To write variables, the ***Preset multiple registers*** (code ***0x10***) command is used. Assuming you have to write to a ***DWORD*** variable allocated in memory at ***MX100.64*** address, this is the formula to calculate the address:

((Address of variable/2)+Offset)-1 → ((64/2)+40000)-1=40031 → 0x9C5F

Being a ***DWORD*** variable, we will write 2 consecutive registers starting from address allocation. Assuming that we need to write the value ***0x12345678*** in the variable, we have:

Modbus RTU frames

Command frame: ***01 10 9C 5F 00 02 04 56 78 12 34 D3 33***
Answer frame: ***01 10 9C 5F 00 02 5F 8A***

Modbus TCP/IP frames

Command frame: ***00 00 00 00 00 0B 01 10 9C 5F 00 02 04 56 78 12 34***
Answer frame: ***00 00 00 00 00 06 01 10 9C 5F 00 02***

The representation of data in SlimLine is ***Little-Endian***. The numbering starts from the least significant byte and ending with the most significant. So as you can see from the response string, the 32-bit value to write ***0x12345678*** is splitted in two 16-bits registers with values ***0x5678***, ***0x1234***.

1.1.4 Access to Real time clock from modbus

It is possible to access to real time clock data using Modbus commands to access registers. The supported functions are:

Code	Function	Tipo oggetto	Tipo accesso	Range indirizzo
03h	Read holding registers	Word (16 Bit)	Read	100-105 (150 for Epoch time)
04h	Read input registers	Word (16 Bit)	Read	100-105 (150 for Epoch time)
06h	Preset single register	Word (16 Bit)	Write	100-105 (150 for Epoch time)
10h	Preset multiple registers	Word (16 Bit)	Write	100-105 (150 for Epoch time)

The registers (16 bits) of the real time clock are allocated in consecutive locations starting from the Modbus address 100. The registers contain the current value of the real time clock and writing a new value, the the real time clock will be automatically updated.

Address	Register	Note
100	Second	Second value (Range from 0 to 59)
101	Minute	Minute value (Range from 0 to 59)
102	Hour	Hour value (Range from 0 to 23)
103	Day	Day value (Range from 1 to 31)
104	Month	Month value (Range from 1 to 12)
105	Year	Year value (Range from 1900 to 2037)

1.1.5 Reading RTC from modbus

To read the values of the real time clock, the ***Read Holding registers*** (code ***0x03***) command is used. We have to read 6 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so ***99*** (***0x0063***).

Modbus RTU frames

Command frame: ***01 03 00 63 00 06 35 D6***
Answer frame: ***01 03 0C 00 1E 00 30 00 0B 00 1D 00 09 07 DA A2 32***

Modbus TCP/IP frames

Command frame: ***00 00 00 00 00 06 01 03 00 63 00 06***
Answer frame: ***00 00 00 00 00 0F 01 03 0C 00 1E 00 30 00 0B 00 1D 00 09 07 DA***

As you can see from the answer, the RTC values is:

Second: 30 (***0x001E***)
Minute: 48 (***0x0030***)
Hour: 11 (***0x000B***)
Day: 29 (***0x001D***)
Month: 9 (***0x0009***)
Year: 2010 (**0x07DA**)

1.1.6 Writing RTC from modbus

To write the values of the real time clock, the ***Preset multiple registers*** (code ***0x10***) command is used. We have to write 6 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so ***99*** (***0x0063***). Assume that we have to set these real time clock values:

Second: 30 (***0x001E***)
Minute: 48 (***0x0030***)
Hour: 11 (***0x000B***)
Day: 29 (***0x001D***)
Month: 9 (***0x0009***)
Year: 2010 (**0x07DA**)

Modbus RTU frames

Command frame: ***01 10 00 63 00 06 08 00 1E 00 30 00 0B 00 1D 00 09 07 DA 5D C8***
Answer frame: ***01 10 00 63 00 06 B0 15***

Modbus TCP/IP frames

Command frame: ***00 00 00 00 00 13 01 10 00 63 00 06 08 00 1E 00 30 00 0B 00 1D 00 09 07 DA***
Answer frame: ***00 00 00 00 00 06 01 10 00 63 00 06***

1.1.7 Epoch time access from modbus

There is also a 32 bits value for date/time Epoch time. The access to this read/write register is always performed using two 16 bits registers.

Address	Register	Note
150	Epoch time	Epoch time

1.1.8 Reading Epoch time from modbus

To read epoch time, the ***Read Holding registers*** (code ***0x03***) command is used. We read 2 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so ***149*** (***0x0095***).

Modbus RTU frames

Command frame: ***01 03 00 95 00 02 D4 27***
Answer frame: ***01 03 04 30 B5 4C A3 90 6C***

Modbus TCP/IP frames

Command frame: ***00 00 00 00 00 06 01 03 00 95 00 02***
Answer frame: ***00 00 00 00 00 07 01 03 04 30 B5 4C A3***

As you can see from the answer, the value is: ***0x4CA330B5 → 1285763253 → GMT: Wed, 29 Sep 2010 12:27:33 UTC***.

1.1.9 Writing Epoch time from modbus

To write the epoch time value, the ***Preset multiple registers*** (code ***0x10***) command is used. We have to write 2 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so ***149*** (***0x0095***). Assume that we have to set these value:

GMT: Wed, 29 Sep 2010 12:27:33 UTC → 1285763253 → 0x4CA330B5

Modbus RTU frames

Command frame: ***01 10 00 95 00 02 04 30 B5 4C A3 50 A3***
Answer frame: ***01 10 00 95 00 02 51 E4***

Modbus TCP/IP frames

Command frame: ***00 00 00 00 00 0B 01 10 00 95 00 02 04 30 B5 4C A3***
Answer frame: ***00 00 00 00 00 06 01 10 00 95 00 02***