Quido - MODBUS

Complete description of MODBUS RTU and TCP protocols in I/O Quido modules

Quido - MODBUS

Datasheet

Created: 9/11/2009

Last update: 3/16/2017 14:00

Number of pages: 16

Address:

Strasnicka 3164
102 00 Prague 10
Czech Republic

Tel:

+420 267 314 267

Fax:

+420 267 314 269

Internet:

www.papouch.com

E-mail:

info@papouch.com

TABLE OF CONTENTS

POPIS

This document describes the MODBUS RTU and MODBUS TCP communication protocols used in I/O Quido modules.

Tip: Detailed information on the MODBUS protocol is available at modbus.org.

Tip: To test communication with Quido via Modbus you can use, for example, ModScan32.

MODBUS RTU: Basic communication parameters

The following parameters apply to Quido modules with RS232 or RS485 interface.

Communication line..................................RS485

Communication speed ....from 110 Bd to 230.4 kBd (*default:* ***9.6 kBd***)

Number of data bits..................................**8**

Parity..no parity, even, odd (*default:* ***no parity***)

Number of stop-bits..................................1, 2 *(default:* ***1****)*

Starting address.......................................0x31

Default protocol (factory settings) ....Spinel
(The way to switch to the MODBUS RTU protocol is shown on the next page.)

MODBUS TCP: Basic communication parameters

In the WEB mode, Guido modules with Ethernet interface can communicate via MODBUS TCP protocol. The communication port can be set in the *Other* tab. The default value of the port is number 502.

List of function codes

The device allows access to its memory - depending on the type of the register – using the following instructions:

- 0x01 .....read coils
- 0x02 .....read discrete inputs
- 0x03 .....read holding registers
- 0x04 .....read input registers
- 0x05 .....set one discrete output[1]
- 0x06 .....set one holding register[1]
- 0x0F .....set multiple discrete output
- 0x10 .....write multiple holding registers
- 0x11 .....identification[1]

[1] This function code can only be used in MODBUS RTU.

SWITCHING BETWEEN PROTOCOLS

The default protocol for Quido RS is Spinel (factory settings). To switch the device to MODBUS use the following instructions in the Spinel protocol. Quido RS can be easily switched to the Modbus protocol (and back) using **ModbusConfigurator**, which is available here:

http://www.papouch.com/en/website/mainmenu/software/modbus-configurator/

Spinel → MODBUS RTU

Enable configuration

Enables a service instruction to be carried out; must immediately precede the “*Switching”* instruction. This instruction cannot be used with the universal or “*Broadcast*” address.

Enquiry:

Instruction code: E4H

Response:

Confirmation code: ACK 00H

Examples:

Enquiry:
2AH,61H,00H,05H,01H,02H,E4H,88H,0DH
Enable configuration.
Response – acknowledgement:
2AH,61H,00H,05H,01H,02H,00H,6CH,0DH

Switching

The protocols are switched by a special instruction in the Spinel protocol, format 97. An address of a particular module must be used the so-called “broadcast” or universal address must not be used). The instruction must be immediately preceded by the “*Enable configuration*” instruction.

Enquiry:

Instruction code: EDH

Response:

Confirmation code: ACK 00H

Examples:

Enquiry:
2AH,61H,00H,06H,66H,02H,EDH,02H,17H,0DH
The command to switch from Spinel to MODBUS RTU protocol.
Response:
2AH,61H,00H,05H,66H,02H,00H,07H,0DH
Confirmation of the command. After sending this response, Guido communicates via MODBUS RTU.

MODBUS RTU → Spinel

It is switched by writing to the Holding register – see page 7.

REGISTERS

Identification of the Device

Reading of the device identification string (Report slave ID).

Function codes:

0x11 – Report slave ID

Parameters:

Number of bytes	1 Byte	according to the string
ID	1 Byte	ID is the same as the device address
RI	1 Byte	Run Indicator – here always 0xFF (switched on)
Data	N Byte	String is the same as in the Spinel protocol. For example: *Quido RS 4/4; v0209.02.27; f66 97; t1*

Holding Register

Device configuration, administration of the counters of pulses and analogue outputs.

Address	*Access*	*Function*	*Description*
0	write	0x06	**Enable Configuration** Writing the 0x00FF value to this memory location must precede all instructions that write in the addresses of 0 to 15 in the holding register. It is used to protect against accidental configuration changes. The *Enable Configuration* instruction must not be written using the 0x10 function code together with other parameters!
1	read, write	0x03, 0x06, 0x10	**Address (ID)**[2] A unique address of the device in the Modbus protocol. A number ranging from 1 to 247 is expected. The address is unique to the Modbus protocol. *The default address is 0x0031.*
2	read, write	0x03, 0x06, 0x10	**Communication speed**[2] The speeds and their corresponding codes: 1 200 Bd......... 0x0003 2 400 Bd......... 0x0004 4 800 Bd......... 0x0005 *9 600 Bd......... 0x0006 (default)* 19 200 Bd......... 0x0007 38 400 Bd......... 0x0008 57 600 Bd......... 0x0009 115 200 Bd......... 0x000A

[2] Writing to this memory location must be preceded by writing the 0x00FF value into the address of 0 in the *Enable Configuration* position. This prevents accidental configuration changes. It is not allowed to write *Enable Configuration* using *Multiply write* simultaneously with other parameters. **After the write is done, the device will be rebooted and counters will be reset to zero.**

<table>
<tr><th><i>Address</i></th><th><i>Access</i></th><th><i>Function</i></th><th><i>Description</i></th></tr>
<tr><td>3</td><td>read, write</td><td>0x03, 0x06, 0x10</td><td><b>Data word</b>[2]
Data word is always eight-bit.
<table>
<tr><th>Value</th><th>Parity</th><th>No of stop-bits</th></tr>
<tr><td><i>0x0000 (default)</i></td><td><i>none (N)</i></td><td><i>1</i></td></tr>
<tr><td>0x0001</td><td>even (E)</td><td>1</td></tr>
<tr><td>0x0002</td><td>odd (O)</td><td>1</td></tr>
<tr><td>0x0003</td><td>none (N)</td><td>2</td></tr>
<tr><td>0x0004</td><td>even (E)</td><td>2</td></tr>
<tr><td>0x0005</td><td>odd (O)</td><td>2</td></tr>
<tr><td>0x0006 to 0x00FF</td><td>none (N)</td><td>1</td></tr>
</table></td></tr>
<tr><td>4</td><td>read, write</td><td>0x03, 0x06, 0x10</td><td><b>Identification of the end of the packet</b>[2]
To configure how long the delay between the bytes must be to be considered the end of the packet. The delay is specified in the number of bytes. You can specify a value ranging from 4 to 100. The default value is 10.</td></tr>
<tr><td>5</td><td>read, write</td><td>0x03, 0x06, 0x10</td><td><b>Communication protocol</b>[2]
Allows the user to switch between communication protocols. After sending the response, the device switches over to the desired protocol. (Each protocol is equipped with an instruction for switching between protocols.)
Code for <i>Spinel: 0x0001 (default)</i>
Code for Modbus RTU: 0x0002</td></tr>
<tr><td>100 – 160</td><td><i>Modbus TCP:</i>
read only
<i>Modbus RTU:</i>
read &amp; write</td><td><i>Modbus TCP:</i>
0x03
<i>Modbus RTU:</i>
0x03, 0x06, 0x10</td><td><b>Status of counters</b>
<i>A counter makes it possible to count changes in the state of an input, meaning changes in the logical state (or state of the connected contact). Each input has its own counter. Number one is added to the counter value for selected changes in the appropriate input (change from 1 to 0, the change from 0 to 1, or both changes).</i>
Here, the current states of the 16-bit counters of all inputs are stored. (Counting is disabled by default.) The total number of registers corresponds to the number of Quido inputs. Thus, in Quido with ten inputs, ten 16-bit registers will be used. The maximum number of the counters is 60 (for any other inputs, counters are not available.)
To reset the counters, write zero. Counters are also reset to zero when the device is powered down or rebooted.
The recommended procedure for continuous reading of the current state of the counters:
1) Read the values of the register <i>Status of counters</i>.
2) Subtract the read value using the following set of registers <i>Deduction from counter</i>.
By using this procedure you will not lose any record of changes in the inputs.</td></tr>
</table>

Address	*Access*	*Function*	*Description*
200 – 260	write	0x06, 0x10	**Deduction from counter** Subtracts the specified value from the current state of the counter. (The value to be deducted shall not be greater than the current state of the counter.) Multiply write (0x10): Unable to write more than 12 registers at once.
300 – 360	read, write	0x03, 0x06, 0x10	**Configuration of the counter** 0.....the counter of this input is <u>disabled</u> 1.....the counter adds one to its value at the <u>leading edge</u> of each signal recorded at the input 2.....the counter adds one to its value at the <u>falling edge</u> of each signal recorded at the input 3.....the counter adds one to its value at <u>any</u> edge (leading and falling) of each signal recorded at the input
500 – 532	read, write	0x03, 0x06, 0x10 (write only one output at same time!)	**Setting the <u>one</u> output for a given period** Activates the <u>one</u> output for a certain period of time - on the selected output, an impulse is triggered with the desired polarity for a specified period of time. The impulse is started immediately upon receipt of this instruction. It is possible to re-trigger the impulse when the previous has not been finished yet. **Upper byte** 0xFF → closed 0x00 → opened **Lower byte** 0x00 to 0xFF → The period during which the output is to be closed or opened (according to the upper byte). The unit is 0.5 sec. It is therefore possible to set a value from 0.5 to 127.5 sec.

<table>
<tr><th>Address</th><th>Access</th><th>Function</th><th>Description</th></tr>
<tr><td>600 – 728</td><td>read, write</td><td>0x03, 0x10
Only all four registers can be read (at once) / written when reading or writing them!</td><td><b>Thermostat</b>
Four consecutive registers are applicable to four outputs and their thermostat settings. First output has a register number 600 and the fourth has register number 603.
<table>
<tr><th>Register</th><th>Meaning</th></tr>
<tr><td>First</td><td>Lower byte format: <b>FSSKTTTT</b>
„<b>F</b>“ – ON/OFF – temperature watching function for Output (OUTx); (1 = ON; 0 = OFF)
„<b>SS</b>“ – Action to perform at a set temperature
00 = switch output ON
01 = switch output OFF
10 = switch output ON for a given time (“positive pulse”)
11 = switch output OFF for a given time (“negative pulse”)
„<b>K</b>“ – Critical temperature tendency – applicable only with pulse action:
0 – temperature rise
1 – temperature drop
„<b>TTTT</b>“ – Binary number (address) of the thermometer to which the following temperature limits apply.</td></tr>
<tr><td>Second</td><td>Temp. format: signed int – upper threshold.</td></tr>
<tr><td>Third</td><td>Temp. format: signed int – lower threshold.</td></tr>
<tr><td>Fourth</td><td>Upper Byte:
Time to switch relay in seconds for pulse action.
Lower byte:
Determines the action upon sensor cable disconnection or damage.
0 – leave output as is
1 – switch output OFF
2 – switch output ON</td></tr>
</table></td></tr>
<tr><td>800 – 832
(Modbus TCP only!)</td><td>read, write</td><td>0x03, 0x10
Only all four registers can be read (at once) / written when reading or writing them!</td><td><b>Temperature watching</b>
Device can watch two temperature limits. If the temperature is outside these limits, it can send a message (as. HTTP GET). Four consecutive registers are applicable to four thermometers. First thermometer has register number 800, last one has number 803.
<table>
<tr><th>Register</th><th>Meaning</th></tr>
<tr><td>First</td><td>Turn ON (0001H) or turn OFF (0000H) temperature watching</td></tr>
<tr><td>Second</td><td>If temperature is out of limits and recurring warning is required, enter interval in seconds.</td></tr>
<tr><td>Third</td><td>Upper temperature limit as an Integer (signed int). It is a temperature multiplied by 10.
Example: For 24.6°C enter 246.</td></tr>
<tr><td>Fourth</td><td>Lower temperature limit as an Integer.</td></tr>
</table></td></tr>
</table>

Input register

Address	*Access*	*Function*	*Description*
0	read	0x04	**Status of the measured temperature** 0.....the value is valid 1.....sensor error or disconnected sensor
1	read	0x04	**Measured value** – an integer The measured temperature as a signed integer. The number shows the measured temperature multiplied by ten. *Example:* The temperature of 23.4 °C is shown in this register as 234.
2, 3	read	0x04	**Measured value** – float The measured temperature as 32 bit float according to IEEE 754. [3]

Discrete Inputs

Function code **0x02** is used for **reading the status of inputs**. It reads 1 to X inputs (maximum according to the number of inputs in Quido). The query specifies the number of the first read input and the number of inputs to be read. The inputs are numbered from zero. For example, the inputs 1 to 10 have serial numbers 0 to 9.

The response contains the states of the inputs represented by individual bits. The value of 1 means an active input (voltage is applied or closed contact), 0 identifies an inactive input. The lowest bit of the first byte of the response represents the state of first input that was addressed in the query.

If the number of inputs is not a multiple of eight, the excess bits are filled with zeros.

Address	*Access*	*Function*	*Description*
0	read	0x02	**Status of the first** required input
1	read	0x02	**Status of the second** required input
…			

Example:

Reading of inputs 1 to 8.

Query:	
Function code	0x02
Address MSB	0x00
Address LSB	0x00
Number of inputs MSB	0x00
Number of inputs LSB	0x08

Response:	
Function code	0x02
Function code	0x01
State of inputs	0xA7

The result of the query is the byte 0xA7, which is 1010 0111 in the binary code. Individual bits correspond to the states of the inputs. The lowest bit represents the input number 1.

[3] Description of the IEEE 754 Standard is available, for example, here: http://en.wikipedia.org/wiki/IEEE_754

Coils

Access to current states of the output relays and their control.

Function code 0x01

This function code is used for **reading the status of outputs**. It reads 1 to X outputs (maximum according to the number of outputs in Quido). The query specifies the number of the first read output and the number of outputs to be read. The outputs are numbered from zero. For example, the outputs 1 to 10 have serial numbers 0 to 9.

The response contains the states of the outputs represented by individual bits. The value of 1 means a closed output, 0 identifies an open output. The lowest bit of the first byte of the response represents the state of first output that was addressed in the query.

If the number of outputs is not a multiple of eight, the excess bits are filled with zeros.

Function codes 0x05 and 0x0F

These function codes have been designed to **control the outputs**. The query specifies the outputs to be set. The outputs are numbered from zero. Thus, for example, output 5 has a serial number 4.

Logical 1 means a closed output, logical 0 an open output.

The response contains the function code, address and number of outputs that have been changed.

Address	*Access*	*Function*	*Description*
0	read, write	0x01, 0x05, 0x0F	**The first** addressed output
1	read, write	0x01, 0x05, 0x0F	**The second** addressed output
…			

Example of reading:

Reading of outputs 1 and 2.

Query:	
Function code	0x01
Address MSB	0x00
Address LSB	0x00
Number of outputs MSB	0x00
Number of outputs LSB	0x02

Response:	
Function code	0x01
Number of bytes	0x01
State of outputs	0x02

The result of the query is the byte 0x02, which is 0000 0010 in the binary code. The second lowest bit is set. The output 1 is open, output 2 is closed. (The remaining bits are filled with zeros.)

Example of writing:

Example of writing the status of outputs 20 to 29 (ten outputs):

The data for the outputs are stored in two bytes: 0xCD and 0x01 (1100 1101 0000 0001 binary)

Bit:	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1
Output No:	27	26	25	24	23	22	21	20	-	-	-	-	-	-	29	28

First, the 0xCD byte is sent with the status of outputs 27 to 20. The lowest bit represents the lowest output 20. The next byte (0x01) contains the remaining bits 28 and 29. The remaining bits are filled with zeros.

Query:	
Function code	0x0F
Address MSB	0x00
Address LSB	0x13
Number of outputs MSB	0x00
Number of outputs LSB	0x0A
Number of bytes	0x02
Values MSB	0xCD
Values LSB	0x01

Response:	
Function code	0x0F
Address MSB	0x00
Address LSB	0x13
Number of outputs MSB	0x00
Number of outputs LSB	0x0A

The result is a change in the state of some outputs in Quido.

Papouch s.r.o.

Data transmission in industry, line and protocol conversions, RS232/485/422/USB/Ethernet/GPRS/ WiFi, measurement modules, intelligent temperature sensors, I/O modules, and custom-made electronic applications.

Address:

Strasnicka 3164
102 00 Prague 10
Czech Republic

Tel:

+420 267 314 267

Fax:

+420 267 314 269

Internet:

www.papouch.com

E-mail:

info@papouch.com