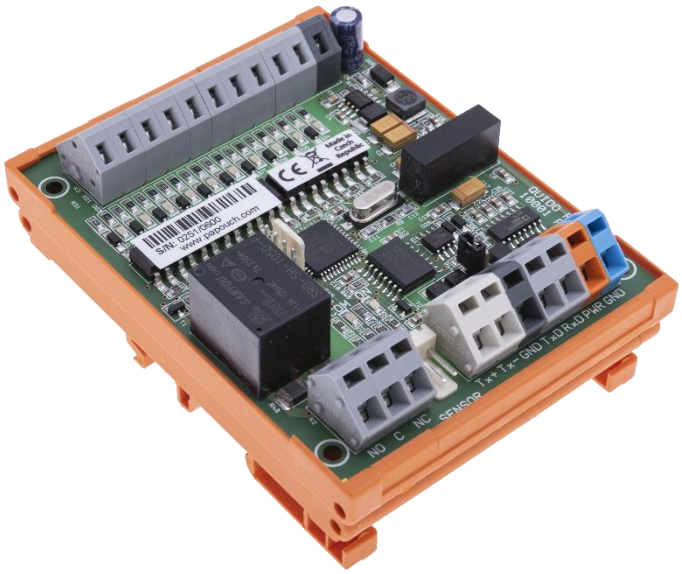
Quido - MODBUS

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



Quido - MODBUS

Datasheet

Created: 9/11/2009

Last update: 2/10/2023 13:48

Number of pages: 16

Papouch s.r.o.

Address:

Strasnicka 3164
102 00 Prague 10
Czech Republic

Phone:

+420 267 314 267

Web:

en.papouch.com

Mail:

info@papouch.com

TABLE OF CONTENTS

Overview of changes in this document

Version 4.50 [1]

- Addressing possible also by serial number.
- Description of configuration jumpers on the board.
- New register Run time with device run time.
- Added measured temperature statuses.
- Added registers with number of inputs, outputs and thermometers.
- Thermometers, input and output statuses are now also available in the Holding register.

[1] By version is meant *[hw-version].[sw-version]* as it can be loaded using function Device Identification (0x11). I.e. a device with identification *Quido RS 4/4; v0209.0**4.50**; f66 97; t1* is version **4.50**.

MODBUS IN QUIDO I/O MODULES

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

- Hardware documentation and feature descriptions are available at papouch.com.
- Detailed information about the MODBUS protocol is available at modbus.org.
- You can use, for example, ModScan32 to test communication with Quido via Modbus.

Communication parameters

Quido RS and Quido USB

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

Quido RS baudrate .....from 1200 Bd to 230.4 kBd (default: 9.6 kBd)

Quido USB baudrate.....115.2 kBd (fixed)

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 (how to switch to Modbus see page 6)

Quido ETH

Quido with Ethernet interface can communicate in WEB mode using MODBUS TCP protocol. Communication port is configurable on tab *Other*. Default port number is 502.

Configuration jumpers

Configuration jumpers are gradually appearing on Quids from 2020 after hardware revisions, making it easier to use Quids in some typical situations.

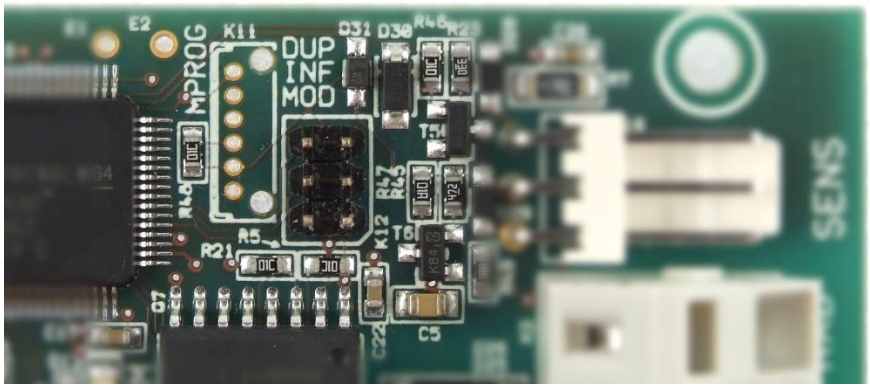


fig. 1 - example of jumpers on Quido RS 2/2

There are three jumpers – Duplex, Info and Modbus. (As you can see on the picture, sometimes is the label in the print slightly shortened.)

Jumper Modbus

If this jumper is shorted when the power is turned on, Quido communicates using the Modbus protocol regardless of the software configuration.

Jumper Info

If this jumper is momentarily shorted when power is connected, Quido sends the current communication parameter settings to the serial port.[2] This information is always sent in Spinel protocol. The RS versions sends the information at 9600 Bd, the USB and ETH versions at 115.2 kBd.

Quido sends first a response to the *Name and Version* instruction and then a packet with address, speed and protocol in ASCII format. Example:

```
*a?"4N?Address:34 Speed:6 Protocol:1ü?
```

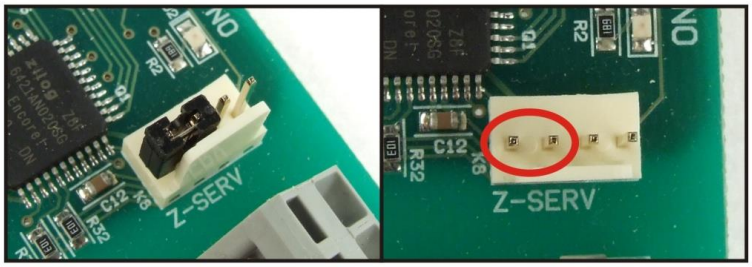
The address is hexadecimal, baudrate is the code according to instruction *Communication Parameters*, and protocol is the protocol number according to instruction *Switch Communication Protocol*.

The Info jumper must not be short-circuited when starting or restarting the device!

Jumper Duplex

This jumper activates the bidirectional transfer mode of input and output status between two Quids 4/4 or 8/8. The kit can be ordered as QuidoDuplexRS or QuidoDuplexETH. The manuals for these kits provide further information on how to configure this mode.

[2] On earlier versions of Quido, the same action can be triggered by shorting two pins on the Z-SERV connector. The pin locations on older Quids can be seen in the picture:



SWITCHING BETWEEN PROTOCOLS

Default protocol for Quido RS and USB is Spinel (factory settings). To switch the device to Modbus RTU use the following instructions in the Spinel protocol. Quido RS can be easily switched to the Modbus protocol (and back) using Modbus configurator (on the same page there are also examples for protocol switching in Python) or by shorting Modbus jumper.

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.

Write

Structure:	→ **0xE4**
Example:	→ 2A 61 00 05 31 02 **E4** 58 0D ← 2A 61 00 05 31 02 **00** 3C 0D

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.

Parameters

protocol	1 byte	Protocol identification number • 0x01: Spinel, format 97 (binary) and 66 (ascii) • 0x02: Modbus RTU (only for Quido RS and Quido USB) • 0x0A: Spinel, format 97 (binary) only

Write

Structure:	→ **0x2A**, protocol
Example:	→ 2A 61 00 06 31 02 **ED 02** 4C 0D • 0x02: Command to switch to the Modbus RTU protocol. ← 2A 61 00 05 31 02 **00** 3C 0D

MODBUS RTU → Spinel

It is switched by writing to the Holding register 0x0005 – see page 9.

REGISTERS

Address

- 0x31: Default device address (decadic 49). Address can be changed at register 1 in Holding register.
- 0x00: Universal Modbus RTU protocol address (decadic 0). If the device receives this address, the instruction is executed but the device does not respond.
- 0xF8: Universal device address (decadic 248). If the device receives this address, the instruction is executed and the device responds. This is only applicable if only one device is connected to the communication line!

How do change address using a serial number? (Modbus RTU only)

With the following procedure it is possible to connect multiple devices with the same address to the RS485 line and then change the address individually:

1) Make a note of device serial number. It is on the label on the device in the format *1395/0069*.The number before the slash is the *Product Type* and the number after the slash is the *Item Number*.
2) Using function code 0x10 and universal address 0xF8, write the following Holding registers to the device in one step:
    a. *Product Type* (addr. 10) – enter the Product type from label.
    b. *Item Number* (addr. 11) – enter the Item Number from label.
    c. *Address* (addr. 12) – enter the new address.
3) The device now communicates with the new address.

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 [3]
- 0x06 .....set one holding register [3]
- 0x0F .....set multiple discrete output
- 0x10 .....write multiple holding registers
- 0x11 .....identification [3]

Device Identification

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

Function codes:

0x11 – Report slave ID [3]

Parameters:

Number of bytes	1 Byte	according to the string

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

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; v0209.04.50; f66 97; t1*

Holding Register

Address	*Access*	*Function*	*Description*
0 [4]	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. This instruction must not be written using the 0x10 function code together with other parameters!
1	read write	0x03 0x06 [3] 0x10	**Address (ID)** [5, 6] 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 [3] 0x10	**Communication speed** [5, 6] 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
3	read write	0x03 0x06 [3] 0x10	**Data word** [5, 6] Data word is always eight-bit.

Value	Parity	No of stop-bits
0x0000 (default)	*none (N)*	*1*
0x0001	even (E)	1
0x0002	odd (O)	1
0x0003	none (N)	2
0x0004	even (E)	2
0x0005	odd (O)	2
0x0006 to 0x00FF	none (N)	1

[4] Sometimes it is possible to see the numbering of registers starting from one, because this first register has address 0.

[5] 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.**

[6] These settings are stored in FLASH memory. This means that (1) the setting is remembered even after the power is turned off, and (2) the number of writes to the memory is limited (typically only a few tens of thousands of writes).

Address	*Access*	*Function*	*Description*
4	read write	0x03 0x06 [3] 0x10	**Identification of the end of the packet** [5, 6] 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.
5	read write	0x03 0x06 [3] 0x10	**Communication protocol** [5, 6] 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 *Spinel: 0x0001 (default)* Code for Modbus RTU: 0x0002 If the Modbus jumper on the electronics board is shorted, the device always communicates via Modbus, regardless of the status of this register!
10	read write	*Only in RTU!* 0x03 0x10 [7]	**Product type** Device type number.
11	read write	*Only in RTU!* 0x03 0x10 [7]	**Item number** Unique item number.
12	read write	*Only in RTU!* 0x03 0x10 [7]	**Address** See How do change address using a serial number? (Modbus RTU only) on page 7.
13	read	*Only in RTU!* 0x03	**Number of inputs**
14	read	*Only in RTU!* 0x03	**Number of outputs**
15	read	*Only in RTU!* 0x03	**Number of thermometers**

[7] Registry 10 až 12 je nutné zapisovat najednou. Zápis nepřepíše hodnoty registrů Typ produktu a Číslo kusu. Zápis do těchto registrů slouží pouze pro funkci nastavení adresy zařízení pomocí sériového čísla (viz str. 6).

Address	*Access*	*Function*	*Description*
100 – 160	*Modbus TCP:* read *Modbus RTU:* read write	*Modbus TCP:* 0x03 *Modbus RTU:* 0x03 0x06 0x10	**Counter status** *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).* 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 a value from this register. 2. Subtract the obtained counter value from the counter using the following register *Subtract value from counter*. Using this procedure, you will not lose any change on input.
200 – 260	write	0x06 [3] 0x10	**Subtract value from counter** Address 200 is input IN1. 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 [3] 0x10	**Counter configuration** At address 300 is counter IN1. • 0: counter is disabled • 1: counter increments at each recorded rising edge on input • 2: counter increments at each recorded falling edge on input • 3: counter increments at any recorded edge (rising or falling) on input

<table>
<tr><th>Address</th><th>Access</th><th>Function</th><th>Description</th></tr>
<tr><td>500
–
532</td><td>read
write</td><td>0x03
0x06 [3]
0x10

using 0x10 only one output!</td><td>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.</td></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>Thermostat
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: FSSKTTTT
„F“ – ON/OFF – temperature watching function for Output (OUTx); (1 = ON; 0 = OFF)
„SS“ – 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”)
„K“ – Critical temperature tendency – applicable only with pulse action:
0 – temperature rise
1 – temperature drop
„TTTT“ – 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><u>Upper Byte</u>:
Time to switch relay in seconds for pulse action.
<u>Lower byte</u>:
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>
</table>

<table>
<tr><th>Address</th><th>Access</th><th>Function</th><th>Description</th></tr>
<tr><td>800
–
832</td><td>read
write</td><td>Only in TCP!
0x03
0x10

Only all four registers can be read (at once) / written when reading or writing them!</td><td>Temperature watching
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>
<tr><td>900
–
904</td><td>read</td><td>0x03</td><td>Copy of input registers 0 to 4</td></tr>
<tr><td>from
1000</td><td>read</td><td>0x03</td><td>Copy of Discrete Inputs
The number of registers corresponds to the number of inputs. The first is IN1. 0x0001 means active input.</td></tr>
<tr><td>from
1200</td><td>read</td><td>0x03</td><td>Copy of Coils
The number of registers corresponds to the number of outputs. The first is OUT1. 0x0001 means active output.</td></tr>
<tr><td>1400
1401</td><td>read</td><td>0x03</td><td>Run time
Time from device power-up or reset in seconds.</td></tr>
</table>

Input register

Address	*Access*	*Function*	*Description*
0 [8]	read	0x04	**Status of the measured temperature** 0 .... the value is valid 1 .... sensor error or disconnected sensor 2 .... temperature is above the upper limit 3 .... temperature is below the lower limit A sensor error or a disconnected sensor will appear after about ten seconds.
1	read	0x04	**Measured value** – signed integer The measured temperature as a signed integer. The number shows the measured temperature multiplied by ten. If the temperature is not valid (according to the status), there is a number -9999. *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. [9] If the temperature is not valid (according to the status), there is a number -9999.
4	read	*Only in RTU!* 0x04	**Temperature unit** 0 .... degrees Celsia 1 .... degrees Fahrenheit 2 .... Kelvin

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
…			

[8] Sometimes it is possible to see the numbering of registers starting from one, because this first register has address 0.

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

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.

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 [3] 0x0F	**The first** addressed output
1	read write	0x01 0x05 [3] 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.

Industrial data transmission, line and protocol converters, RS232, RS485, RS422, USB, Bluetooth, Ethernet, LTE, WiFi, measurement modules, smart temperature sensors, I/O modules, custom development and manufacturing.

Address:

Strasnicka 3164
102 00 Prague 10
Czech Republic

Phone:

+420 267 314 267

Web:

en.papouch.com

Mail:

info@papouch.com