RESI-C4-xxx
RESI-T4-xxx
IoT Controller

based on
Raspberry Pi® Compute Module 4 and PI4
Our series of intelligent IoT controller

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
More information under www.raspberrypi.org

Content

1 Our portfolio

Based on the Raspberry Pi® PI4 and Compute Module 4 LINUX board, we offer a broad spectrum of IoT Controller.

1.1 RESI-T4-xxx Compact IoT Controller

Our RESI-T4 series is based on the standard Raspberry Pi® PI4 board. We packed the board into an industrial housing for the DIN rail and we extended the controller with various features:

- Industrial grade power supply 12-48Vdc
- Dimension XT4: 72x110x62mm (WxHxD)
- SD-CARD Slot with 32GB SD CARD for LINUX and your software
- Versions with integrated serial interfaces: RS232 or RS485
- Versions with integrated KNX interface
- Versions with integrated CAN 2.0 or CAN FD interface
- ARM Co-Processor with real time clock with backup capacitor, ferromagnetic RAM for permanent data storage, unique serial number, Status LEDs and 8 pin DIP switch

The modules are designed for mounting on a DIN EN50022 rail. But the modules offer also a wall mounting option.

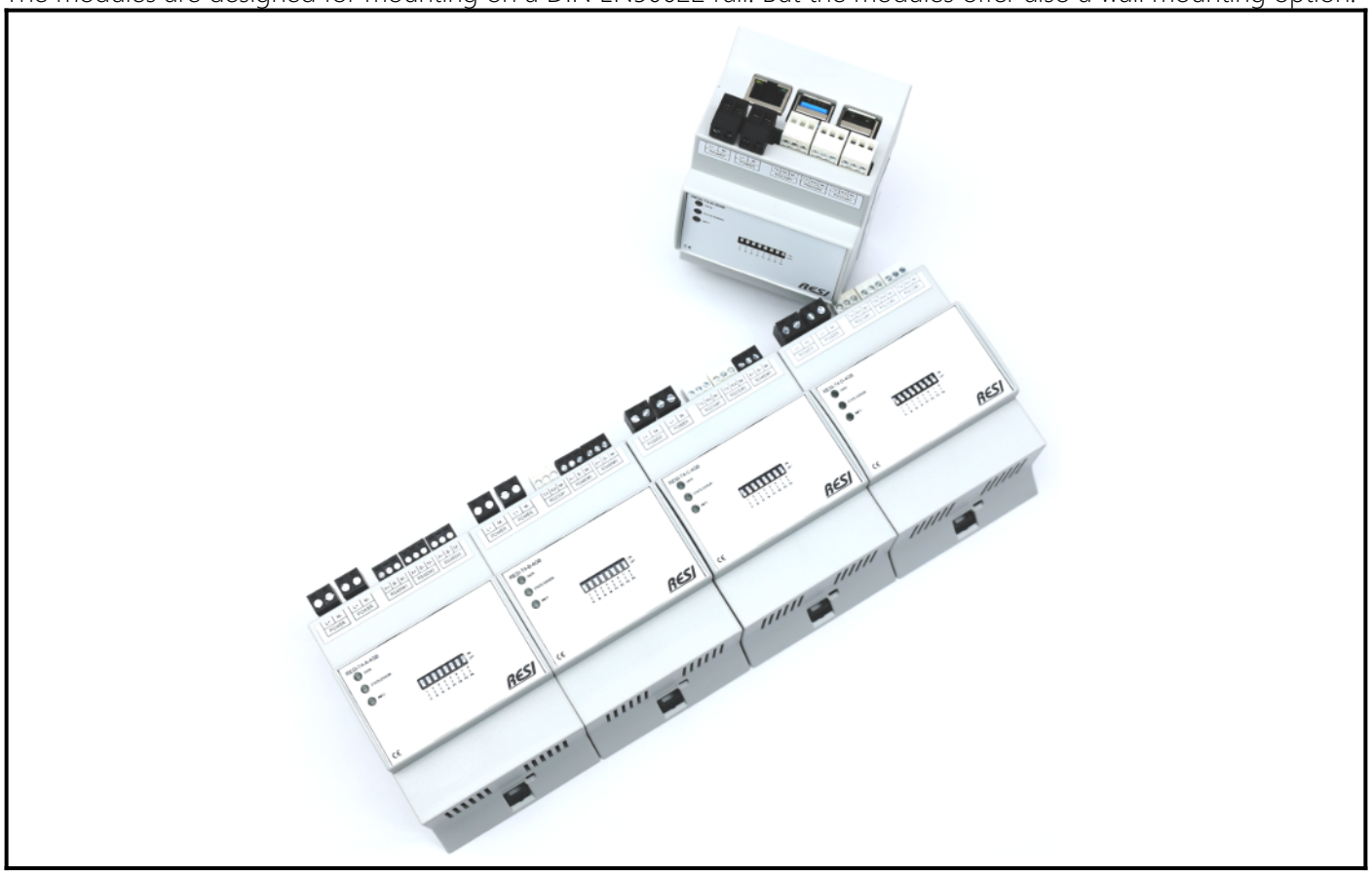


Figure: Our series of RESI-T4-xxx IoT Controller

1.2 RESI-C4-xxx IoT Controller with integrated IOs

Our RESI-C4 series is based on the Raspberry Pi® Compute Module 4 board. We build around the CPU module an industrial grade controller with integrated IOs. The main features of this series are:

- Industrial grade power supply 12-48Vdc
- Dimension XT4: 72x110x62mm (WxHxD) or XT8 143x110x62mm or XT12 213x110x62mm
- SD-CARD Slot with 32GB SD CARD for LINUX and your software
- One serial interface RS485
- Versions with integrated LTE modem
- Versions with second Ethernet
- ARM Co-Processor for management of the integrated IOs.
- Up to 152 integrated IOs.
- ARM Co-Processor with real time clock with backup capacitor, ferromagnetic RAM for permanent data storage, unique serial number, Status LEDs and 8 pin DIP switch

The modules are designed for mounting on a DIN EN50022 rail. But the modules offer also a wall mounting option.

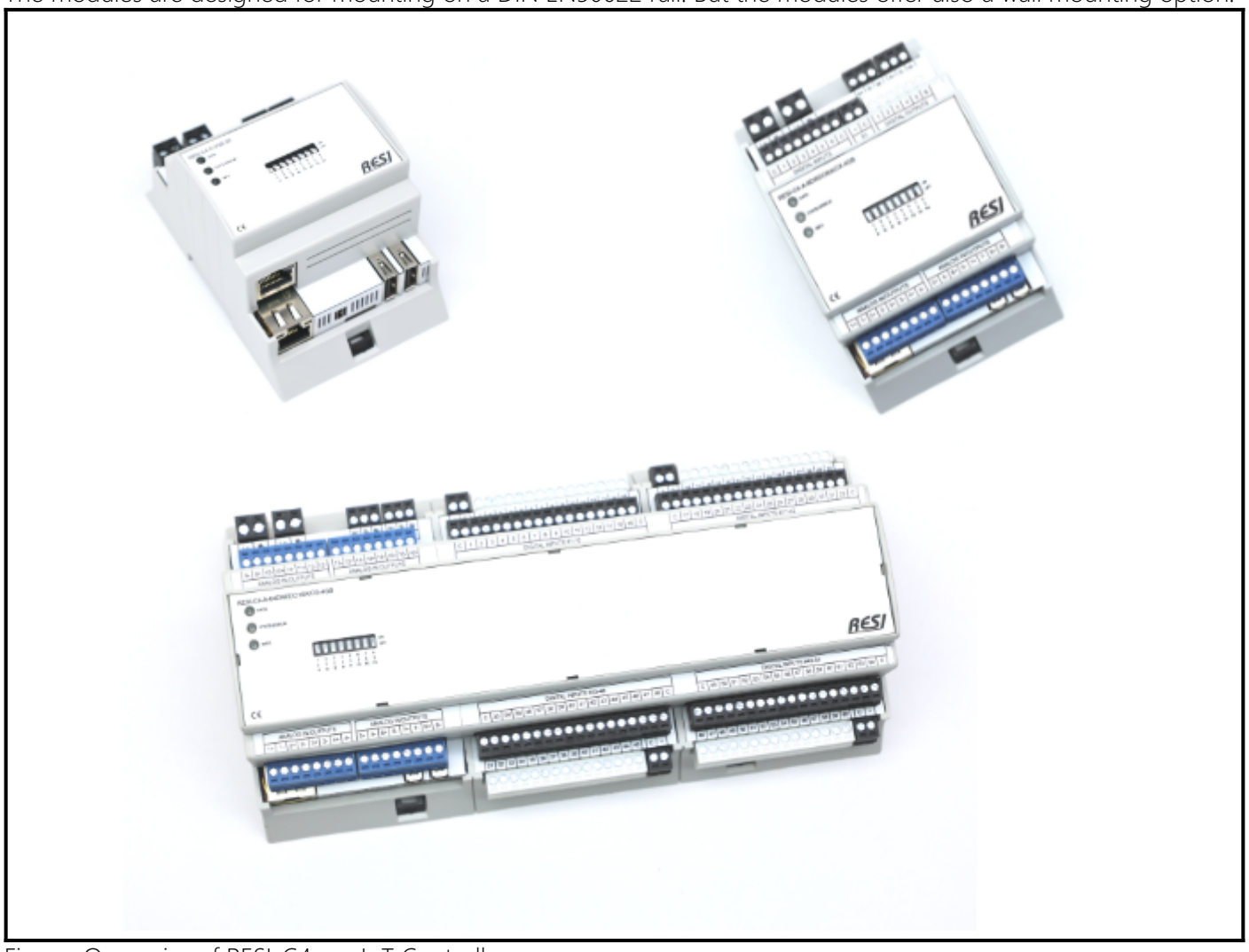


Figure: Our series of RESI-C4-xxx IoT Controller

2 Declaration of conformity

2.1 CE

All products have passed the CE tests for environmental specifications when shielded cables are used for external wiring. We recommend the use of shielded cables.

2.2 Safety instructions

Danger to life through electrical current!

Only skilled personal trained in electro-engineering should perform the described steps in the following chapters. Please observe the country specific rules and standards. Do not perform any electrical work while the device is connected to power.
Pay attention to the following rules:

1. Disconnect the system from power
2. Secure the system against automatic power on
3. Check that the system is de-energized
4. Cover other energized parts of the system

IMPORTANT HINT: Before you start with the installation and the initial setup of the device, you have to read this document and the attached installation guide and the actual manual for the device very carefully. You have to follow all the herein given information very accurate!

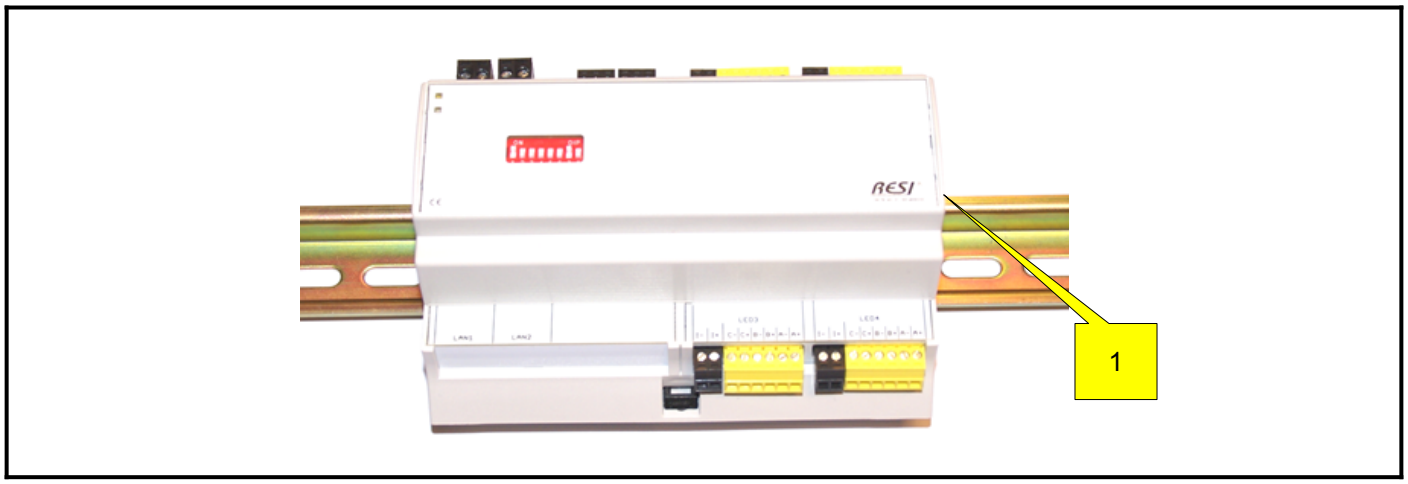
- ❑ Only authorized and qualified personnel are allowed to install and setup the device!
- ❑ The connection of the device must be done in de-energized state!
- ❑ Do not perform any electrical work while the device is connected to power!
- ❑ Disable and secure the system against any automatic restart or power on procedure!
- ❑ The device must be operated with the defined voltage level!
- ❑ Supply voltage jitters must not exceed the technical specifications and tolerances given in the technical manuals for the product. If you do not obey this issue, the proper performance of the device cannot be guaranteed. This can lead to fail functions of the device and in worst case to a complete breakdown of the device!
- ❑ You have to obey the current EMC regulations for wiring!
- ❑ All signal, control and supply voltage cables must be wired in a way, that no inductive or capacitive interference or any other severe electrical noise disturbance may interfere with the device. Wrong wiring can lead to a malfunction of the device!
- ❑ For signal or sensor cables you have to use shielded cables, to avoid damages through induction!
- ❑ You have to obey and to apply the current safety regulations given by the ÖVE, VDE, the countries, their control authorities, the TÜV or the local energy supply company!
- ❑ Obey country-specific laws and standards!
- ❑ The device must be used for the intended purpose of the manufacturer!
- ❑ No warranties or liabilities will be accepted for defects and damages resulting from improper or incorrect usage of the device!
- ❑ Subsequent damages, which results from faults of this device, are excluded from warranty and liability!
- ❑ Only the technical data, wiring diagrams and operation instructions, which are part to the product shipment are valid!
- ❑ The information on our homepage, in our data sheets, in our manuals, in our catalogs or published by our partners can deviate from the product documentation and is not necessarily always actual, due to constant improvement of our products for technical progress!
- ❑ In case of modification of our devices made by the user, all warranty and liability claims are lost!
- ❑ The installation has to fulfil the technical conditions and specifications (e.g. operating temperatures, power supply, ...) given in the devices documentation!
- ❑ Operating our device close to equipment, which do not comply with EMC directives, can influence the functionality of our device, leading to malfunction or in worst case to a breakdown of our device!
- ❑ Our devices must not be used for monitoring applications, which solely serve the purpose of protecting persons against hazards or injury, or as an emergency stop switch for systems or machinery, or for any other similar safety-relevant purposes!
- ❑ Dimensions of the enclosures or enclosures accessories may show slight tolerances on the specifications provided in these instructions!
- ❑ Modifications of this documentation is not allowed!
- ❑ In case of a complaint, only complete devices returned in original packing will be accepted!

3 Mounting for XT4, XT8 or XT12

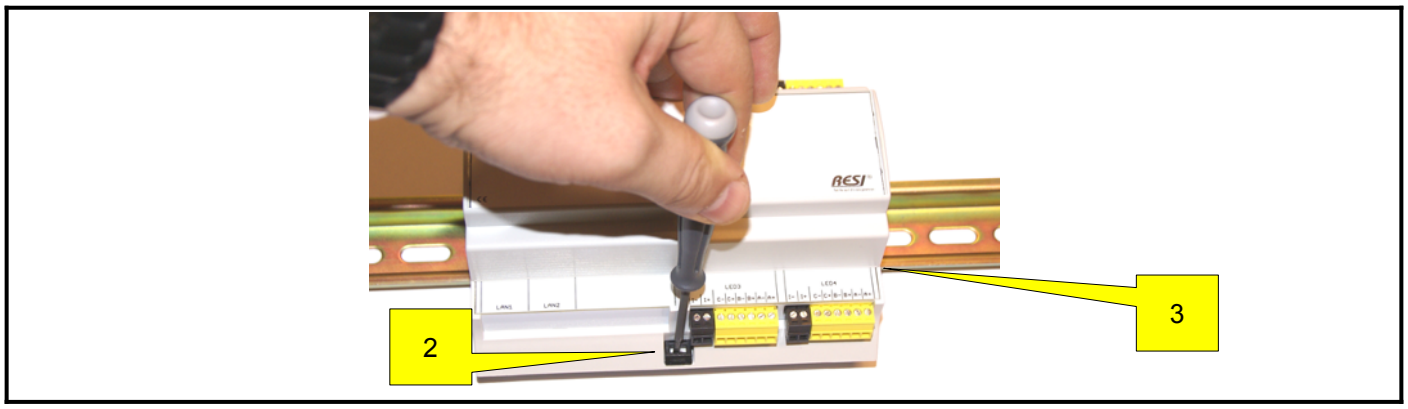
Our IoT controllers are designed for mounting onto a 35mm DIN-EN50022 rail or for wall mounting.
Please note, that in the following mounting description we use only symbolic photos of our IoT controllers.

3.1 Mounting on a DIN EN50022 rail

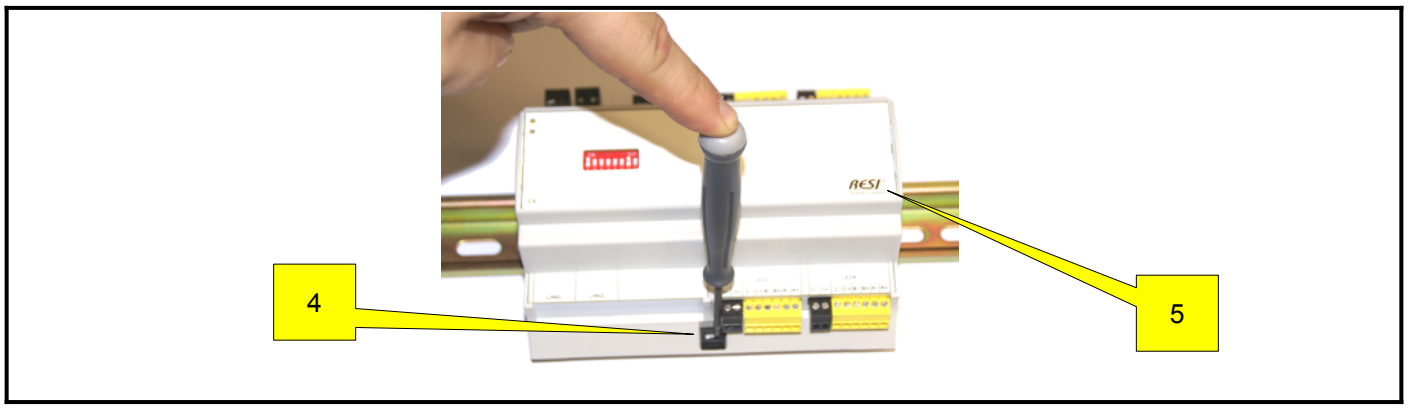
First snap in the top part of the module into the DIN rail (1). The bottom part of the module is not snapped into the DIN rail at this moment.



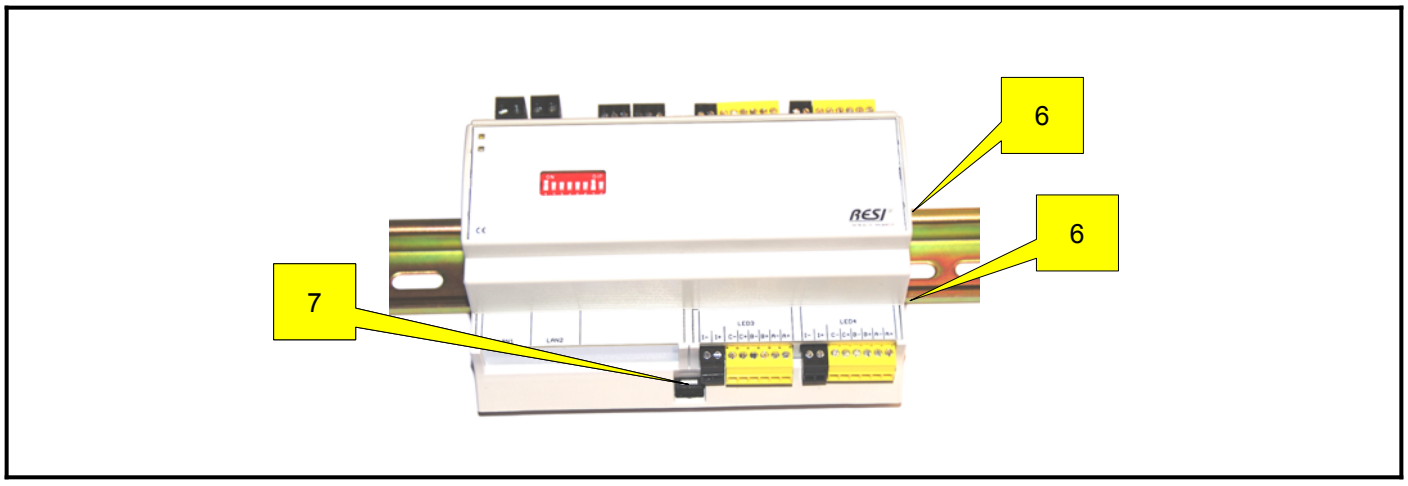
Then open the black hook with a screw driver (2). Now press the module with the opened hook onto the DIN rail until both sides of the module snap into the DIN rail (3). Release the screw driver now. The hook snaps into the DIN rail and the module is now mounted correctly onto the DIN rail.



To remove the module from the DIN rail, you must open the hook with a screwdriver first. (4). Afterwards tilt the bottom side of the module upwards with the open hook (5). Now remove the module slightly from the DIN rail with the top side, to completely hang out the module from the DIN rail.

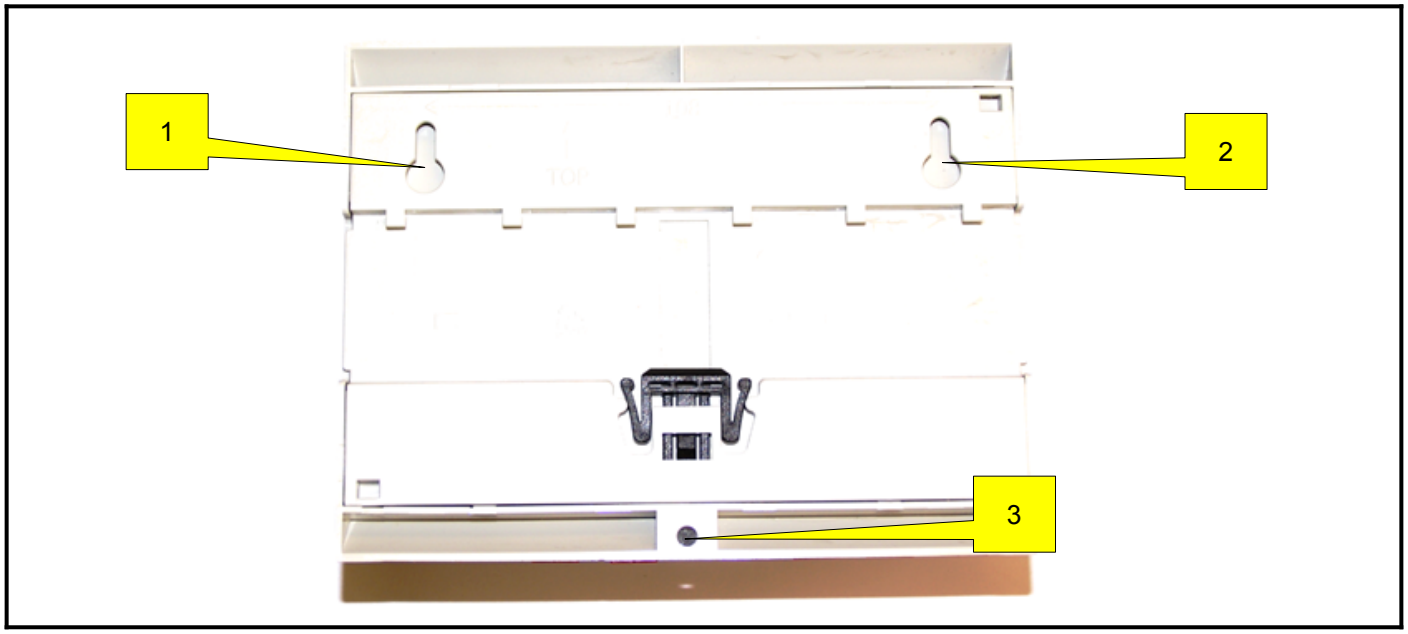


The module is correctly mounted, if the module has snapped into the DIN rail on both sides of the housing (6) and if the hook has snapped in too (7).

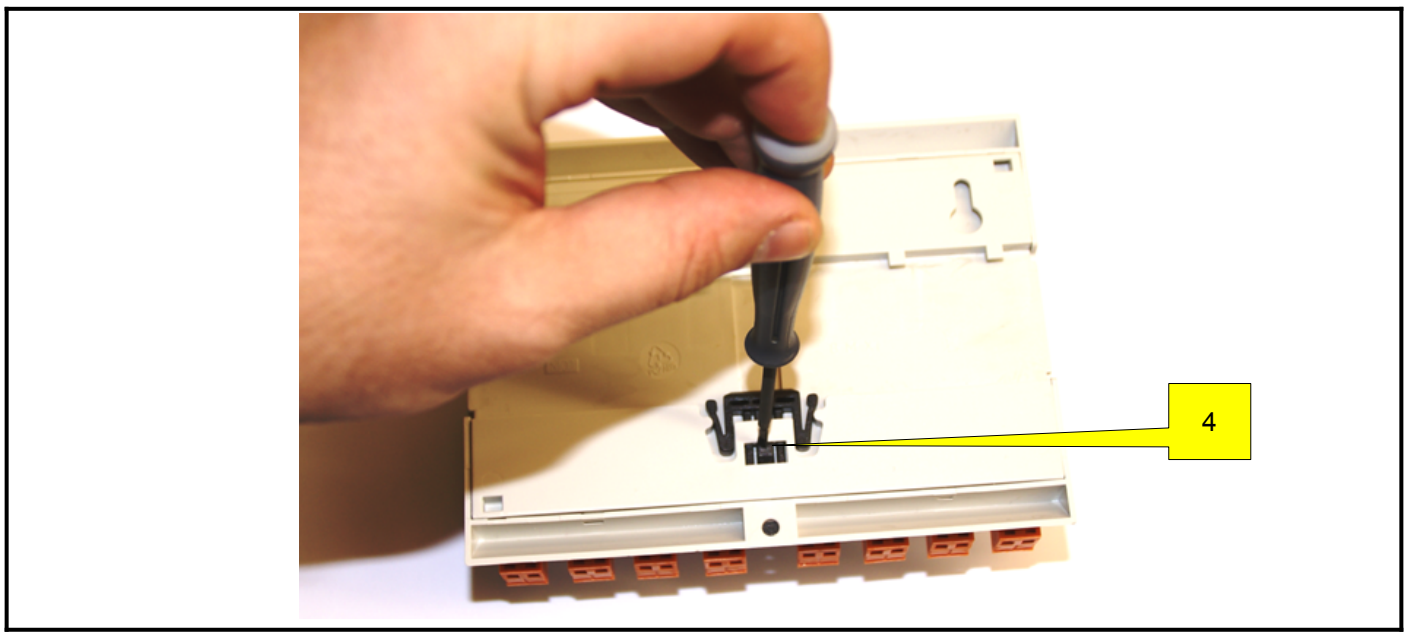


3.2 Mounting onto a wall

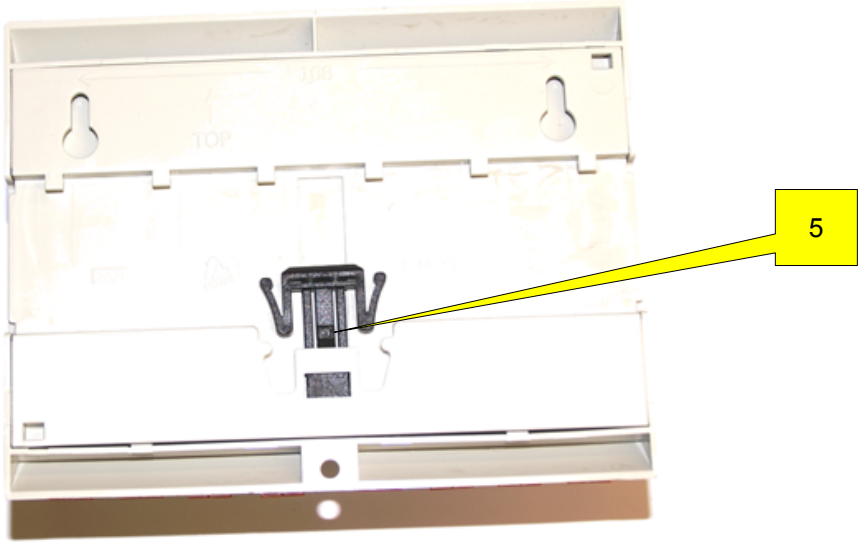
Our modules can also be mounted onto a wall. Turn over the module as shown in the picture below:



You will notice, that there are two holes for wall hooks or screws on the top side of the housing. (1) and (2). On the bottom side you will notice a small hole for a screw to fix the housing on the wall from the front (3). But first we have to remove the hook, which blocks the screw hole in the housing.



Press carefully the screwdriver onto the hook to open the lock (4) and pull back the hook to the inner side of the housing bottom to remove the hook. If the hook is not snapped into the housing, you can remove the hook by hand (5) and the screw hole for fixing the housing with a screen from the front side of the housing (6).



4 General technical data

In this section you will find all technical data which is common to all IO modules. In the specific sections of the individual IO modules you will find only the differences and extensions to this standard description.

4.1 RESI-T4-xxx Basic technical data

Power supply	
Supply voltage	12-48 V = +/- 10%
Power consumption	see individual technical data for specific IoT controller

Raspberry Pi 4® module	
Module type	Raspberry PI 4 with 2/4/8GB RAM
	More details on the official Raspberry homepage

Operating system	LINUX
FLASH	in SD-CARD slot: 32GB

Serial interfaces	
Up to three serial interfaces:	RS232 or RS485

Ethernet interface	
Cable connection	via RJ 45 socket

USB interface	1xUSB 2.0, 2xUSB 3.0

HDMI interface	2xHDMI micro connectors 4K

AUDIO+VIDEO interface	1xAUDIO+VIDEO out connector

Real-Time-Clock	Yes, with external backup capacitor

General	
Storage temperature	-20...85 °C
Operating temperature	0...50 °C
Humidity	25...90% r.H. non-condensing
Protection class	IP20 (EN 60529)
Dimensions LxWxH	see section Dimension
Weight	see individual technical data for specific IO module
Installation	on DIN EN50022 rail and on wall

Approvals	
CE conformity	Yes

4.2 RESI-C4-xxx Basic technical data

Power supply	
Supply voltage	12-48 V = +/- 10%
Power consumption	see individual technical data for specific IoT controller
Raspberry Pi 4® module	
Module type	Raspberry PI Compute Module 4 with 2/4/8GB RAM
	More details on the official Raspberry homepage
Operating system	LINUX
FLASH	in SD-CARD slot: 32GB
Serial interfaces	1xRS485
Ethernet interface	1xEthernet or 2xEthernet
Cable connection	via RJ 45 socket
USB interface	2xUSB 2.0
HDMI interface	1xHDMI micro connector 4K
Real-Time-Clock	Yes, with external backup capacitor
General	
Storage temperature	-20…85 °C
Operating temperature	0…50 °C
Humidity	25…90% r.H. non-condensing
Protection class	IP20 (EN 60529)
Dimensions LxWxH	see section Dimension
Weight	see individual technical data for specific IO module
Installation	on DIN EN50022 rail and on wall
Approvals	
CE conformity	Yes

4.3 RESI-T4-xxx: Basic terminals

The RESI-T4-xxx IoT controller come in a housing with removable clamps. All T4 IoT Controller offer the following terminals:

L+, M-	Power supply via two separated plug-in 2-pin terminal blocks.	
	For daisy chain IN and OUT power supply of many modules	
	Pin 1:	L+: 12-48 V=
	Pin 2:	M-: Ground
	Terminal type:	RM5
Depending on IoT controller:		Up to 3xRS485 or RS232 or up to 2xRS232 or RS485 and 1xKNX
RS485#?	RS485 serial interface	
	Pin 1:	A+: RS485 DATA+ signal
	Pin 2:	B-: RS485 DATA- signal
	Pin 3:	M-: RS485 ground signal
	Terminal type:	RM3.5
RS232#?	RS232 serial interface	
	Pin 1:	TX: RS232 DATA+ signal
	Pin 2:	RX: RS232 DATA- signal
	Pin 3:	M-: RS232 ground signal
	Terminal type:	RM3.5

KNX	KNX interface	
	Pin 1:	K+: KNX+ signal (RED)
	Pin 2:	K-: KNX- signal (BLACK)
	Terminal type:	RM3.5
CAN/CAN FD	CAN 2.0 or CAN FD interface	
	Pin 1:	H: CAN HIGH signal
	Pin 2:	L: CAN LOW signal
	Pin 3:	G: CAN Ground signal
	Terminal type:	RM3.5
Terminal type RM5	Cable cross section:	max. 2.5 mm², max. 14AWG
	Screw:	M3
	Tightening torque:	max. 0.5Nm, max. 4.43 Lb-in
Terminal type RM3.5	Cable cross section:	max. 1.5 mm², max. 16AWG
	Screw:	M2
	Tightening torque:	max. 0.2Nm, max. 1.77 Lb-in

4.4 RESI-C4-xxx: Basic terminals

The RESI-C4-xxx IoT controller come in a housing with removable clamps. All C4 IoT Controller offer the following terminals:

L+, M-	Power supply via two separated plug-in 2-pin terminal blocks.	
	For daisy chain IN and OUT power supply of many modules	
	Pin 1:	L+: 12-48 V=
	Pin 2:	M-: Ground
	Terminal type:	RM5
SIO1	RS485 serial interface	
	Pin 1:	A+: RS485 DATA+ signal
	Pin 2:	B-: RS485 DATA- signal
	Pin 3:	M-: RS485 ground signal
	Terminal type:	RM3.5
Terminal type RM5	Cable cross section:	max. 2.5 mm², max. 14AWG
	Screw:	M3
	Tightening torque:	max. 0.5Nm, max. 4.43 Lb-in
Terminal type RM3.5	Cable cross section:	max. 1.5 mm², max. 16AWG
	Screw:	M2
	Tightening torque:	max. 0.2Nm, max. 1.77 Lb-in

4.5 MODBUS and ASCII commands

Our controllers offer the possibility to communicate with the internal ARM Co-processor either via ASCII commands or with MODBUS RTU master protocol.

Our **RESI-T4-xxx** controller offer **ASCII and MODBUS RTU master** communication via **dev/ttyACM0**
Our **RESI-C4-xxx** controller offer **ASCII** communication via **dev/ttyACM0** and **MODBUS RTU maste**r communication via **dev/ttyACM1**

4.5.1 MODBUS mapping+ASCII command list for T4+C4 IoT controller

Please refer to the external document for detailed documentation of the current MODBUS+ASCII commands for this IoT controller. You will find it on our website www.RESI.cc in the document section for the specific IoT controller.

IMPORTANT HINT:
The ASCII commands and answers may vary through the actual amount of IOs of your IoT controller model. So in this document we show ASCII commands only as a hint, how to use the commands.

The MODBUS register indices are not always the same for all IoT controllers. So be aware that in this documentation we only give you a sample MODBUS register or coil out of any IoT controller of our portfolio to show the basic register mapping for a function.

So take this document only as a hint, how to read or write to the registers and coils. But the correct index is only found in the current document for your IoT Controller. This document with the list of all ASCII commands is found on our web server. It has the name **RESI-L-<ControllerName>-MODBUS+ASCII-EN.pdf**

4.5.2 MODBUS RTU master communication

For communication with the ARM Co-processor, the LINUX software can use MODBUS/RTU master protocol.
The Co-processor offers the following MODBUS functions:

- READ COILS (function code: 1)
- READ CONTACTS (function code: 2)
- WRITE SINGLE COIL (function code: 5)
- WRITE MULTIPLE COILS (function code: 15)

- READ HOLDING REGISTER (function code: 3)
- READ INPUT REGISTER (function code: 4)
- PRESET SINGLE REGISTER (function code: 6)
- PRESET MULTIPLE REGISTERS (function code: 16)

IMPORTANT:
The internal Co-processor uses **always UnitID 1** because it is the only MODBUS/RTU slave on this serial line!
The settings of the baud rate, parity, stop bits are irrelevant, due to the fact that the Co-processor is physically connected via USB to the LINUX system.

HINT:
The functions READ HOLDING REGISTER, READ INPUT REGISTER and PRESET MULTIPLE REGISTERS are restricted to max. 125 register per request.
The functions READ COILS, READ CONTACTS and WRITE MULTIPLE COILS are restricted to max. 1000 bits per request.

4.5.3 HOWTO map values to MODBUS registers

MODBUS is an international standard for communication between host systems like PLCs, DDCs or Industrial PCs and peripheral components or sensors.

More details about the MODBUS standard and the MODBUS protocol can be found here:
http://en.wikipedia.org/wiki/Modbus
http://www.modbus.org/

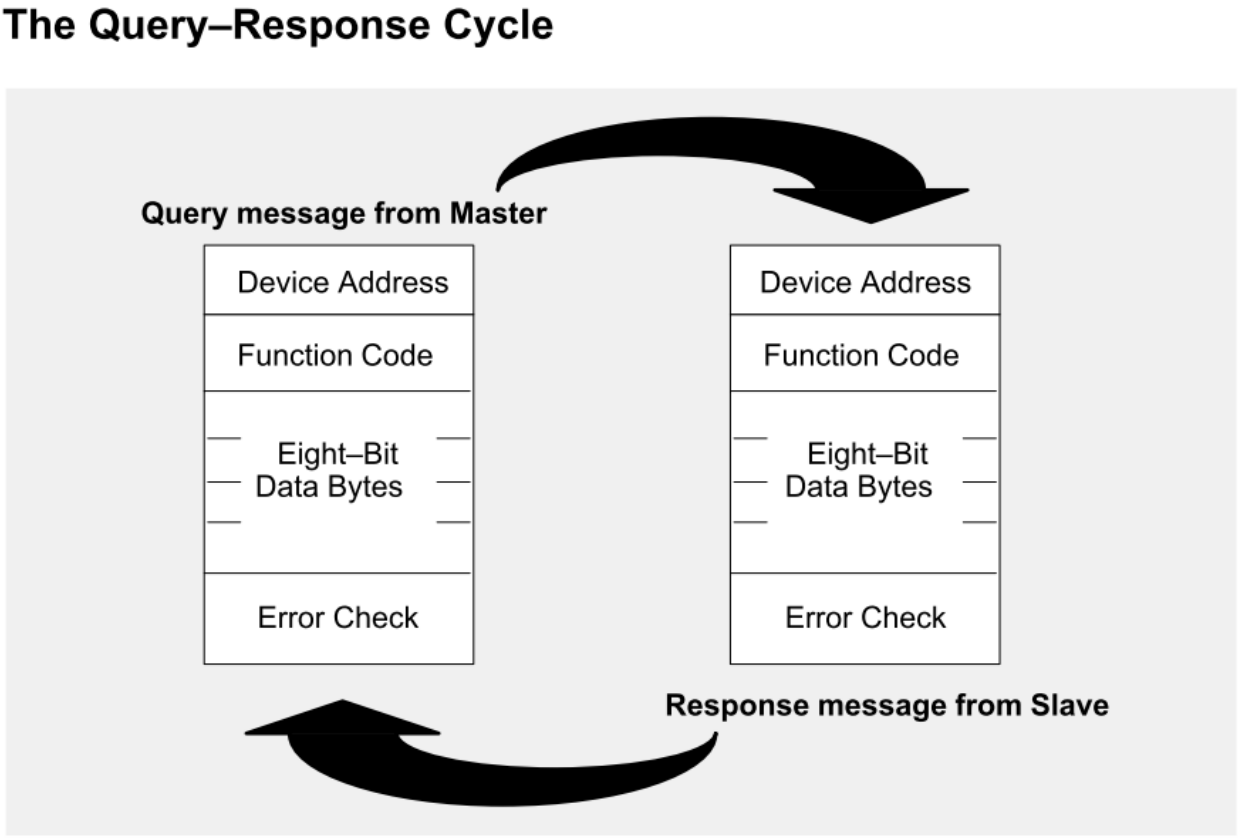
You can find a documentation about this in the internet called "PI_MBUS_300.pdf" , which describes the MODBUS protocol pretty good.

There are three different MODBUS protocol versions available:
MODBUS/TCP: Used for communication with TCP/IP systems
MODBUS/RTU: A binary version of the MODBUS protocol
MODBUS/ASCII: An ASCII text based version of the protocol

To communicate with our ARM Co-processor you have to use MODBUS/RTU master protocol version.

4.5.4 MODBUS query response cycle

MODBUS is a master slave protocol. This means, the master (your host system) has to send a protocol to a specific MODBUS slave (one of our converters), then this specific slave answers to the master, and then the master asks the next slave. The address of the slave is the so-called device address or unit address, which we mentioned before. See the below graphic, how a basic MODBUS request and response cycle looks like.



4.5.5 MODBUS/RTU telegram structure

A MODBUS/RTU protocol frame consists out of the following fields:

START: There is no specific start character, so a pause of four character timings depending on the baud rate of your communication must be established. This means at least for four characters, that there must be no communication on the serial line!

ADDRESS: This is the unit address of the slave, the master wants to talk to. It' s a number between 0 and 255.

FUNCTION: This defines the type of data communication, the master wants to handle with the slave. Refer to the next pages for a detailed description of the functions.

DATA: This is a block of individual data bytes.

CRC CHECK: This is the checksum, to let the master and slave check, if the received protocol is correct and without communication errors.

END: Same as the start condition. Again there must not be communicated for at least 4 character times on the serial line.

IMPORTANT HINT: If there is more than one MODBUS slave on a serial line, the pausing of the START and END sequence are essential to re synchronize the slaves in case of data loss. If the host doesn' t keep this gaps, communication with the slaves can be corrupted or impossible!

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1–T2–T3–T4	8 BITS	8 BITS	*n* x 8 BITS	16 BITS	T1–T2–T3–T4

4.6 MODBUS commands

The MODBUS standard defines many available commands . But not all systems handle the complete spectrum of telegrams. Our converter handles only all telegrams necessary for using holding and INPUT registers.

We support
01 READ COIL STATUS
02 READ INPUT STATUS
03 READ HOLDING REGISTER
04 READ INPUT REGISTER
05 FORCE SINGLE COIL
06 PRESET SINGLE REGISTER
15 FORCE MULTIPLE COILS
16 PRESET MULTIPLE REGISTER

IMPORTANT HINT: All other protocols are ignored by our converters.

So what are COILS or INPUTS ?

According to the MODBUS standard, a MODBUS/RTU slave can hold up to 65535 coils and 65535 inputs. Each coil or input is a 1 bit register, capable for binary values between 0 and 1.

A MODBUS/RTU master system can read and write the contents of those registers with the functions:
01 READ COIL STATUS
02 READ INPUT STATUS
05 FORCE SINGLE COIL
15 FORCE MULTIPLE COILS

Our Co-processor has only one table for coils and inputs. So it makes no difference if your read coils or inputs. You will read the same state!

So what are HOLDING or INPUT REGISTERs ?

According to the MODBUS standard, a MODBUS/RTU slave can hold up to 65535 HOLDING registers and 65535 INPUT registers. Each holding or input register is a 16 bit register, capable for integer values between 0 and 65535 or in hexadecimal from 0x0000 to 0xFFFF.

A MODBUS/RTU master system can read and write the contents of those registers:
03 READ HOLDING REGISTER
04 READ INPUT REGISTER
06 PRESET SINGLE REGISTER
16 PRESET MULTIPLE REGISTER

Our Co-processor has only one table for holding and input registers. So it makes no difference if your read holding registers or input register. You will read the same state!

IMPORTANT HINT:

A MODBUS/RTU master can read and write into this registers with a 16 bit index, called the starting address. The problem is the definition of the starting address. A 16 bit value can store the values from 0 to 65535. But according the MODBUS standard the registers are numbered from 1 to 65536. So, if the MODBUS standard talks about register 1, an index of 0 must be used as start address in the telegram. You have to check carefully, how this index is interpreted by the manufacturer's documentation.

Code	Name
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
07	Read Exception Status
08	Diagnostics
09	Program 484
10	Poll 484
11	Fetch Comm. Event Ctr.
12	Fetch Comm. Event Log
13	Program Controller
14	Poll Controller
15	Force Multiple Coils
16	Preset Multiple Registers
17	Report Slave ID
18	Program 884/M84
19	Reset Comm. Link
20	Read General Reference
21	Write General Reference

Whenever you get a description of registers for a MODBUS device, the first question to solve is: How is the enumeration of the registers done?! Does the author use base=0, then he talks about the real start index of the telegram. Does the author mean base=1, conforming to naming conventions of the MODBUS consortium, then you have to subtract 1 before using this address in your telegrams.

IMPORTANT HINT:
If we display a holding register address like 4x00009 in our tool, we assume base=1 conforming to the standard. So your host system has to send the start index 00008 decimal to read out the correct register.

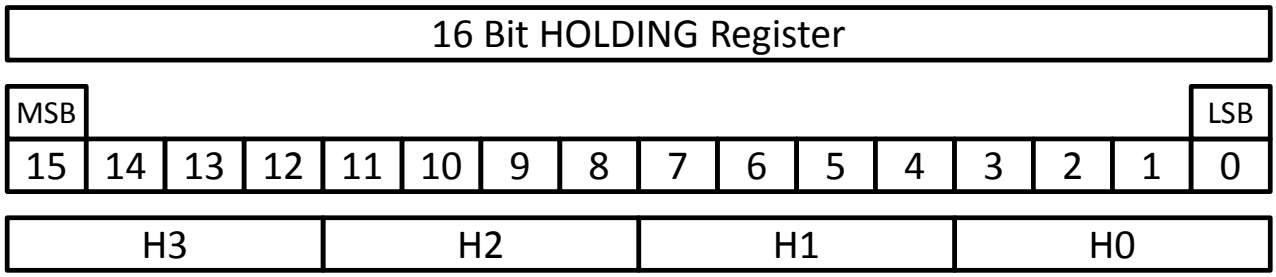
Start Index (Base=0)	MODBUS Register (Base=1)	Description
0	1	The first holding register
1	2	The second holding register
2	3	The third holding register
...	...	
65534	65535	The penultimate holding register
65535	65536	The last holding register

4.6.1 MODBUS 16 bit holding register structure

Here we give a brief introduction, how to build the contents of a MODBUS holding register, and how a hexadecimal writing of a 16 bit register looks like. We assume, that the user is familiar to hexadecimal and binary number systems and also how a computer stores data into its internal memory.

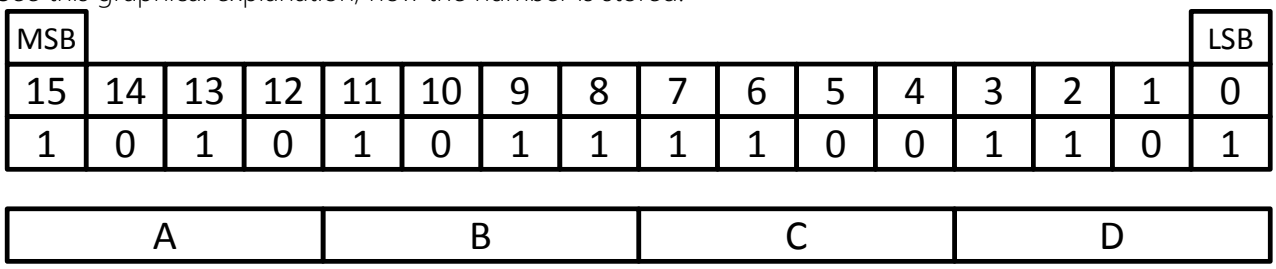
For more details consult the internet:
http://en.wikipedia.org/wiki/Hexadecimal
http://en.wikipedia.org/wiki/Binary_number

Usually a hexadecimal digit describes 4 bits. So we can group the 16 bits into 4 hexadecimal digits named H3,H2,H1,H0. This means eg. the hexadecimal number 0xABCD stands for H3=A, H2=B, H1=C, H0=D.



0xA=1010 binary, 10 dec, 0xB=1011,11 dec, 0xC=1100,12 dec and 0xD=1101, 13 dec. So the resulting binary number is 1010101111001101b or 43981 decimal.

See this graphical explanation, how the number is stored:



4.6.2 MODBUS big vs. least significant byte order

Now the first problem for a host system arises:
If we take the 16 bit number 0xABCD, we have to use 2 bytes to store this value internally. There are two concurrent versions of how to store this value in the RAM:

INTEL byte order, Little endian systems store the least significant byte first. So a memory map for 0xABCD look like:

Memory address 0	CD
Memory address 1	AB

MOTOROLA byte order, Big endian systems store the most significant byte first. So a memory map for 0xABCD look like:

Memory address 0	AB
Memory address 1	CD

Consult the internet for more details about this storage system.

http://en.wikipedia.org/wiki/Endianness

4.6.3 MODBUS storing large data into 16 bit registers

After years, the market found out, that the capabilities of storing only 16 bit numbers into one holding register is not enough for many applications. The most common solution to store more than 16 bit values into holding registers is to use more than one register to hold the value. For storing e.g. a 32 bit value, we use two consecutive 16 bit holding registers, for storing a 32 bit float value we also use also two consecutive 16 bit registers!

We want to store the 32 bit integer value 0x12345678 into two consecutive holding registers starting at 4x00020. The memory map of the holding registers look like:

		16 bit value
Start Index 19	Holding Register 4x00020	0x1234
Start Index 20	Holding Register 4x00021	0x5678

But again, we can also store the reverse word order into two consecutive registers. Then the result looks like this:

		16 bit value
Start Index 19	Holding Register 4x00020	0x5678
Start Index 20	Holding Register 4x00021	0x1234

So none of the above mentioned orders is better than the other. It depends only on the programmer, how the 32 bit value is treated.

Be aware, that both systems (host and converter) have to treat the 32 bit value in the same way. Otherwise you will read out wrong data! We will discuss this issue later in combination with 32 bit float numbers.

Our converter uses the second described way to store 32 bit values. We follow the little endian strategy of INTEL systems and store 0x5678 into the first HOLDING register, and then we store 0x1234 in the consecutive register.

4.6.4 MODBUS datatypes in our Co-processor

Our Co-processor supports the following data types for storing values into MODBUS registers.

16 bit signed binary: This is an integer number between -32767..0..+32768 or 0x0000 to 0xFFFF hex. This number needs exactly one HOLDING register.

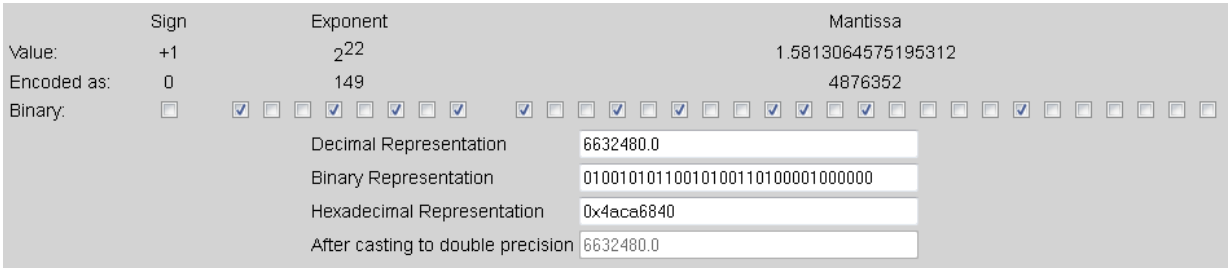
32 bit singed binary: This is an integer number between -2,147,483,647..0..+2,147,483,648 or 0x00000000 to 0xFFFFFFFF hex. This number needs two consecutive holding registers. We store the least significant word first. The serial number 2544082 is in hex 0x26D1D2. This leads to the following HOLDING register layout:

		16 bit value
Start Index 0	Holding Register 4x00001	0xD1D2 or 53714 dec
Start Index 1	Holding Register 4x00002	0x0026 or 38 dec

32 bit IEEE floating point: This is a float number using 32 bit. As before, this float needs two consecutive holding registers. We store the least significant word first. The energy value of 6632480,00 is defined in 32 bit hex with 0x4ACA6840. This leads to the following HOLDING register layout. For more details search in the internet or

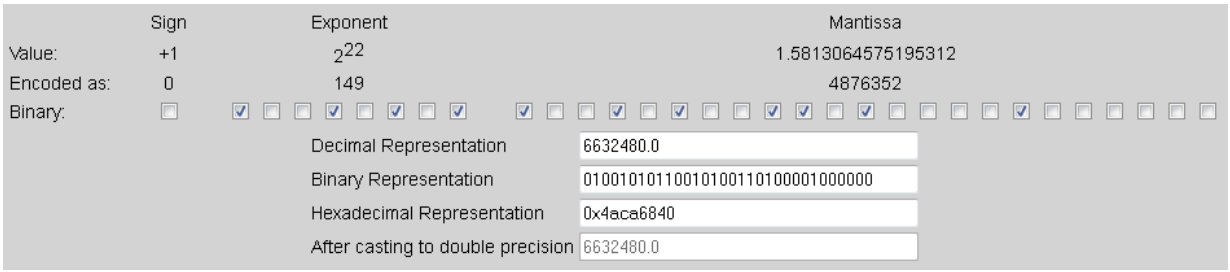
consult http://en.wikipedia.org/wiki/IEEE_floating_point or try out some float values and their hexadecimal representation under http://www.h-schmidt.net/FloatConverter/IEEE754.html

		16 bit value
Start Index 0	Holding Register 4x00001	0x6840
Start Index 1	Holding Register 4x00002	0x4ACA



32 bit IEEE floating point inverse: This is a float number using 32 bit. Again this float needs two consecutive holding registers. We store the least significant word first. The energy value of 6632480,00 is in 32 bit hex 0x4ACA6840. This means the following HOLDING register layout. For more details search in the internet or consult http://en.wikipedia.org/wiki/IEEE_floating_point or try out some float values and their hexadecimal representation under http://www.h-schmidt.net/FloatConverter/IEEE754.html

		16 bit value
Start Index 0	Holding Register 4x00001	0x4ACA
Start Index 1	Holding Register 4x00002	0x6840



IMPORTANT HINT:
32 bit floats are very tricky! Eg. The value 3,5351799 is represented internally as 0x40624063. But the reverse word order (if the host reads out the wrong register indexes or the host corrupts the word order) 0x40634062 leads to the float number 3,5508046. So this error in your software is very hard to find! Be very cautious, which register indexes you read and how the word order of the two registers are interpreted.

32 bit date&time: This is a compressed format using 32 bit. Again the least significant word is stored into the first register. The structure of the 32 bits are:

Bits 0..7: minute
Bits 8..15: hour
Bits 16..20: day
Bits 21..24: month
Bits 25..31: year

The current date & time "07.04.00 01:13" is represented hexadecimal with 0x0087010d (8847628dec) and stored as followed:

		16 bit value
Start Index 0	Holding Register 4x00001	0x010D
Start Index 1	Holding Register 4x00002	0x0087

4.6.5 MODBUS datatype storage and common pitfalls

In general MODBUS uses 16 bit wide registers. So if you use only datatypes, which needs also only one register, the mapping is easy. But as soon as you use datatypes, e.g. UINT32, which need two or more MODBUS registers, you can map the values in different ways.

We do a simple sample. We want to store the 32 bit unsigned integer value in hexadecimal 0x12345678 in MODBUS holding registers starting with index 4x00010. The mapping can be done in two different ways:

MODBUS Register	Storage of UINT32 datatype
4x00010 I:9	The high word of the 32 bit value 0x12345678 is stored in the first 16 bit wide MODBUS register. This means the value 0x1234 is stored here.
4x00011 I:10	The low word of the 32 bit value 0x12345678 is stored in the second 16 bit wide MODBUS register. This means the value 0x5678 is stored here.

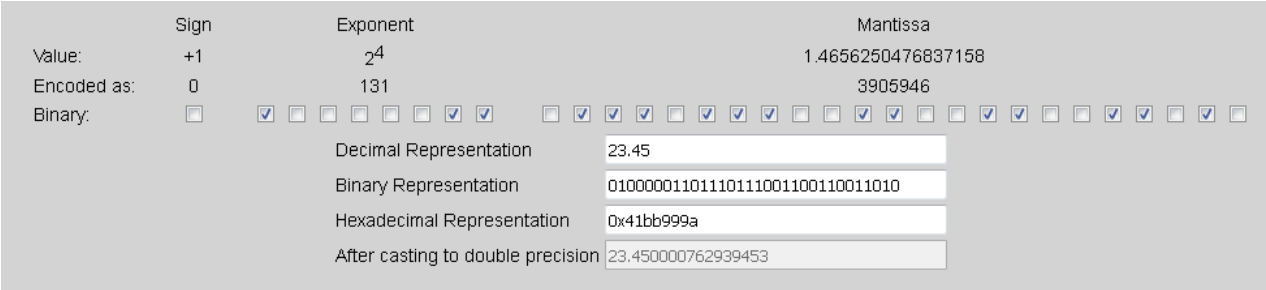
But it is only one possibility, that we store the high word in the first MODBUS register. With the same right, we can define to store the low word in the first register, and the high word in the second.

The result will look like this:

MODBUS Register	Storage of UINT32R datatype
4x00010 I:9	The low word of the 32 bit value 0x12345678 is stored in the first 16 bit wide MODBUS register. This means the value 0x5678 is stored here.
4x00011 I:10	The high word of the 32 bit value 0x12345678 is stored in the second 16 bit wide MODBUS register. This means the value 0x1234 is stored here.

More complicated is the storage of a FLOAT32 value into two consecutive holding registers. We use a standard room temperature e.g. 23,45 °C as a value, we want to store it into two registers.

First we have to translate this value into a valid IEE754 float value. Therefore we use a perfect site in the internet (http://www.h-schmidt.net/FloatConverter/IEEE754.html):



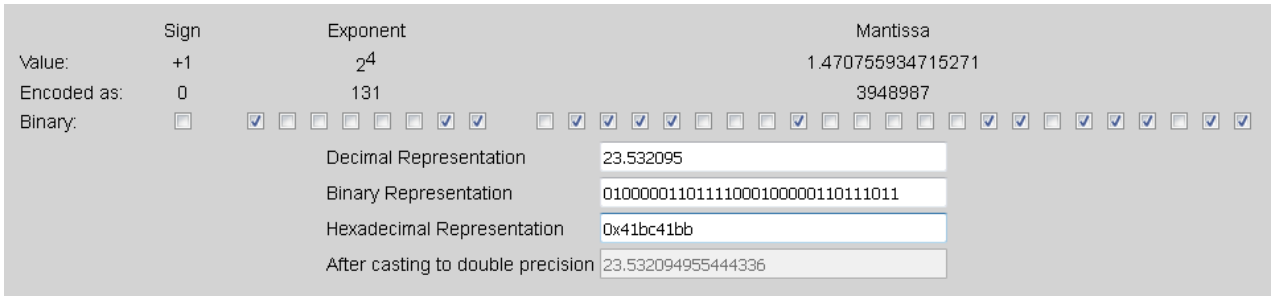
We enter the value 23.45 and we get a 32 bit hexadecimal representation of the float value. It is the number 0x41BB999A. Now we store this value in the same way, we have stored the UINT32 value into two registers:

MODBUS Register	Storage of FLOAT32 datatype
4x00010 I:9	The high word of the 32 bit float value 0x41BB999A is stored in the first 16 bit wide MODBUS register. This means the value 0x41BB is stored here.
4x00011 I:10	The low word of the 32 bit float value 0x41BB999A is stored in the second 16 bit wide MODBUS register. This means the value 0x999A is stored here.

But we can also use the reverse notation:

MODBUS Register	Storage of FLOAT32R datatype
4x00010 I:9	The low word of the 32 bit float value 0x41BB999A is stored in the first 16 bit wide MODBUS register. This means the value 0x999A is stored here.
4x00011 I:10	The high word of the 32 bit float value 0x41BB999A is stored in the second 16 bit wide MODBUS register. This means the value 0x41BB is stored here.

Now we show a common pitfall in writing and reading more than one MODBUS register and rebuilding a value. We use a different float value. In hexadecimal it is 0x41BC41BB. Again we use the online converter:

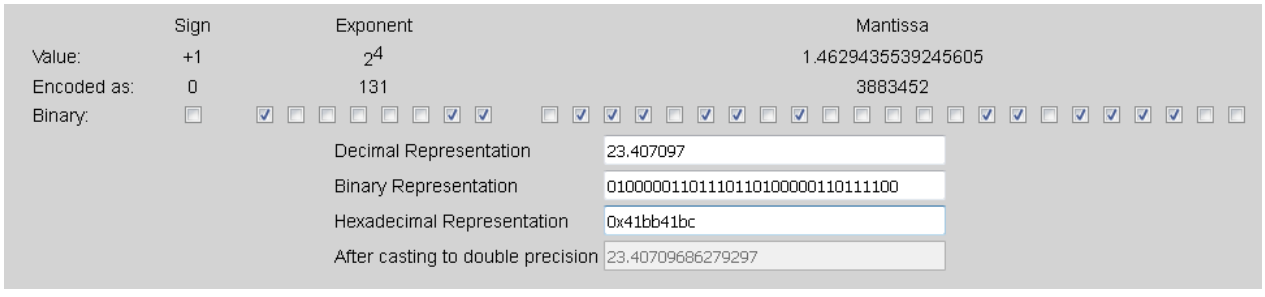


You notice, the float value is 23.532095.

Now we store it with HIGH word first into two registers:

MODBUS Register	Storage of FLOAT32 datatype
4x00010 I:9 HIGH WORD	The high word of the 32 bit float value 0x41BC41BB is stored in the first 16 bit wide MODBUS register. This means the value 0x41BC is stored here.
4x00011 I:10 LOW WORD	The low word of the 32 bit float value 0x41BC41BB is stored in the second 16 bit wide MODBUS register. This means the value 0x41BB is stored here.

But now we make a very big mistake, we read the two registers and restore the hexadecimal value in our host software in the reverse word order. First low word, then high word. The result is the 32 bit value 0x41BB41BC instead the correct value 0x41BC41BB. Then we convert this into an IEE754 float value.



The result is 23.407097. This is not far away from the original number of 23.532095! So this massive software error can be undiscovered for a long time. Only if the reverse float value generates numbers which are physically not possible for the measured signal, this error is discovered!

4.6.6 MODBUS data type table

The following table shows, how more complex data types are stored in successive 16 bit holding or input registers within the MODBUS registers:

MODBUS DATATYPE	SIZE	WORD ORDER	DESCRIPTION
UINT16	16 bits 1 register	none	Defines a 16 bit unsigned integer value in the range of 0 to 65535 or 0x0000 to 0xFFFF
SINT16	16 bits 1 register	none	Defines a 16 bit signed integer value in the range of -32768 to +32767 or 0x8000 to 0x7FFF
UINT32	32 bits 2 register	0:High Word 1:Low Word	Defines a 32 bit unsigned integer value in the range of 0 to 4.294.967.295 or 0x00000000 to 0xFFFFFFFF
SINT32	32 bits 2 register	0:High Word 1:Low Word	Defines a 32 bit signed integer value in the range of −2.147.483.648 to +2.147.483.647or 0x80000000 to 0x7FFFFFFF
UINT32R	32 bits 2 register	0:Low Word 1:High Word	Defines a 32 bit unsigned integer value in the range of 0 to 4.294.967.295 or 0x00000000 to 0xFFFFFFFF with reverse word order
SINT32R	32 bits 2 register	0:Low Word 1:High Word	Defines a 32 bit signed integer value in the range of −2.147.483.648 to +2.147.483.647or 0x80000000 to 0x7FFFFFFF with reverse word order
FLOAT32	32 bits 2 register	0:High Word 1:Low Word	Defines a 32 bit float value in the range of $\pm 1.4 \cdot 10^{-45}$ to $\pm 3.403 \cdot 10^{38}$. A mantissa of 23 bits and an exponent of 8 bits are used. The value can store 7 to 8 digits after the comma.
FLOAT32R	32 bits 2 register	0:Low Word 1:High Word	Defines a 32 bit float value in the range of $\pm 1.4 \cdot 10^{-45}$ to $\pm 3.403 \cdot 10^{38}$. A mantissa of 23 bits and an exponent of 8 bits are used. The value can store 7 to 8 digits after the comma. The two 16 bit words are stored in reverse order.
DOUBLE64	64 bits 4 register	0:Highest Word 1:Higher Word 2:Lower Word 3:Lowest Word	Defines a 64 bit float value in the range of $\pm 4.24 \cdot 10^{-324}$ to $\pm 1{,}798 \cdot 10^{308}$. A mantissa of 52 bits and an exponent of 11 bits are used. The value can store 15 to 16 digits after the comma.
DOUBLE64R	64 bits 4 register	0:Lowest Word 1:Lower Word 2:Higher Word 3:Highest Word	Defines a 64 bit float value in the range of $\pm 4.24 \cdot 10^{-324}$ to $\pm 1{,}798 \cdot 10^{308}$. A mantissa of 52 bits and an exponent of 11 bits are used. The value can store 15 to 16 digits after the comma. The four 16 bit words are stored in reverse order.

4.6.7 MODBUS table

COILS (1x) & **INPUTS** (2x)
The module holds internally a list of 1 bit coil and input register. Those registers can be read by the host with the function READ COIL STATUS (function code: 1). If the register can also be modified by the host, the host can use the functions FORCE SINGLE COIL (function code: 5) and FORCE MULTIPLE COILS (function code: 15).

In addition the SAME registers are also readable over the function READ INPUT STATUS (function code: 2). This is for host systems, which do not support all MODBUS/RTU functions properly.

The MODBUS convention defines 65535 possible coils with the notation 1x00001 to 1x65536. Inputs are usually noted with 2x00001 to 2x65536. Please refer the software MODBUS POLL as a sample for this notation. Internally in the MODBUS/RTU frames an index notation is used, which starts with 0 and ends with 65535. So we decided to note in the following document a register with: 1x00100 for the coil 100, 2x00100 as a hint, that you can read this register also as the input 100, and in addition also the real index of the protocol index 99 with the notation I:99.

HOLDING REGISTER (3x) & INPUT REGISTER (4x)
The module holds internally a list of 16 bit wide holding register. Those registers can be read by the host with the function READ HOLDING REGISTER (function code: 3). If the register can also be modified by the host, the host can use the functions PRESET SINGLE REGISTER (function code: 6) and PRESET MULTIPLE REGISTERS (function code: 16).

In addition the SAME holding registers are also readable over the function READ INPUT REGISTER (function code: 4). This is for host systems, which do not support all MODBUS/RTU functions properly.

The MODBUS convention defines 65535 possible holding register with the notation 4x00001 to 4x65536. Input register are usually noted with 3x00001 to 3x65536. Please refer the software MODBUS POLL as a sample for this notation. Internally in the MODBUS/RTU frames an index notation is used, which starts with 0 and ends with 65535. So we decided to note in the following document a register with: 4x00100 for the holding register 100, 3x00100 as a hint, that you can read this register also as the input register 100, and in addition also the real index of the protocol index 99 with the notation I:99.

RESI Configurator RESI-C4-32DI12RO,32DI12ROxAIOX-V1003 RESI-C4-xxx MB Basics

Register NAME	MODBUS Register	Register VALUE	NEW REAL VALUE	NEW VALUE	DATA TYPE	DO WRITE
PRODUCT INFO						
HW_GROUP	3x65201 4x65201 I:65200	50176,0xC400 B:C4 00			UINT16 R/O	
This is the group of hardware of the current product						
HW_TYPE	3x65202 4x65202 I:65201	1,0x0001 B:00 01			UINT16 R/O	
This is the type of hardware of the current product						
SW_VERSION	3x65203 4x65203 I:65202	272,0x0110 B:01 10			UINT16 R/O	
		SW VERSION:0.1.0				
This is the current software version of the firmware						
SW_AUTHOR	3x65204 4x65204 I:65203	21321,0x5349 B:53 49			UINT16 R/O	
This is the current software author of the firmware						
MANUFACTURER	3x65205 4x65205 I:65204	1380275017,0x52455349 B:52 45 53 49			UINT32 R/O	
This is the current software author of the firmware						
NUMBER OF DIGITAL INPUTS	3x65207 4x65207 I:65206	0,0x0000 B:00 00			UINT16 R/O	
		Number of DIS:0				
This is the current software version of the firmware						
NUMBER OF DIGITAL OUTPUTS	3x65208 4x65208 I:65207	0,0x0000 B:00 00			UINT16 R/O	
		Number of DOS:0				
This is the current software version of the firmware						
NUMBER OF ANALOG INPUTS	3x65209 4x65209 I:65208	0,0x0000 B:00 00			UINT16 R/O	
		Number of AIS:0				
This is the current software version of the firmware						
NUMBER OF ANALOG OUTPUTS	3x65210 4x65210 I:65209	0,0x0000 B:00 00			UINT16 R/O	
		Number of AOS:0				
This is the current software version of the firmware						

4.7 ASCII protocol

All of our IoT controller communicate with very simple ASCII commands with the LINUX software.
The following special characters are used in this description:

stands for the **hash sign** ASCII character 35dec or 0x23
: stands for the **colon** ASCII characters 58dec or 0x3A
= stands for the **equal sign** with the ASCII code 61ec or 0x3D
- stands for the **minus sign** with the ASCII code 45dec or 0x2D
, stands for the **comma** with the ASCII code 44dec or 0x2C
<CR> or $_{CR}$ stands for the **CARRIAGE RETURN** ASCII character 13dec or 0x0D. This is shown as CR in the following.
<SP> or ☐ stands for **SPACE**. This is the space in ASCII code 32dec or 0x20. The space is shown as , hereinafter.
In the following **<ADR>** is used for the **bus address**. This can be transmitted in decimal or hexadecimal and is separated from the following command with a comma (ASCII characters 44dec or 0x2C). Hexadecimal numbers always start with 0x. Only the ASCII characters '0' - '9' 48dec to 57dec, 0x30-0x39 and 'A' to 'F', 65dec to 70dec, 0x41-0x46 may be used.

Our Co-processor uses always 255 or 0xFF as a bus address. Due to the fact, that the Co-processor is the only device on the serial line you can also avoid the bus number.

4.7.1 COMMUNICATION SEQUENCE

In principle, the Co-processor does not send any characters by itself. Communication always starts from the LINUX software.

The command structure looks like this:

The host sends a command or a command with parameters without a bus address:
#<command><CR> or
#<command>:<parameter><CR>

The module responds when it feels addressed with the telegram:
#<respond><CR>

The host sends the following to the module with the bus address (For our Co-processor always 255 or 0xFF):
#<ADR>,<command><CR> or
#<ADR>,<command>:<parameter><CR>

The Co-processor then replies with:
#<ADR>,<reply><CR>

4.7.2 Example: Query VERSION

This command provides the current type of the module.

Host command:
#VERSION<CR> or
#<ADR>,VERSION<CR>

Reply:
#VERSION:<HIGH>.<MED>.<LOW><CR> or
#<ADR>,VERSION:<HIGH>,<MED>,<LOW><CR>

<HIGH>.<MED>.<LOW> represents the current software version, e.g. 3.0.0

Examples:
#VERSION$_{CR}$
#VERSION:3.0.0$_{CR}$

With broadcast address in decimal:
#255,VERSION$_{CR}$
#255,VERSION:3.0.0$_{CR}$

With broadcast address in hexadecimal:
#0xFF,VER$_{CR}$
#255,VERSION:3.0.0$_{CR}$

4.7.3 Example: Query module TYPE

This command provides the current type of the module.

Host command:
#TYPE<CR> or
#<ADR>,TYPE<CR>

Respond:
#TYPE:<TYP><CR> or
#<ADR>,TYPE:<TYP><CR>

<TYP> represents the current type of the module. A RESI-T4-A is shown as an example

Examples:
#TYPE$_{CR}$
#TYPE:RESI-T4-A$_{CR}$

#255,TYP$_{CR}$
#255,TYPE:RESI-T4-A$_{CR}$

4.7.4 Table of all ASCII commands

For every IoT controller you will find an actual list with all ASCII commands on our web server www.RESI.cc. Browse to the product and download the PDF document with all commands.
The file name will be like RESI-L-<ProductName>-MODBUS+ASCII-EN.pdf

Only the version including the bus address is listed here. It has already been explained that this can also be omitted. If an argument has the addition dec, it is returned as a decimal number. If an argument has the addition hex, a hexadecimal number is returned. Many commands return both the decimal and the hexadecimal representation. The host can thus choose which number conversion he would like to carry out.

Please refer to the description of individual products for more details about the available ASCII commands.

RESI Configurator RESI-C4-32DI12RO,32DI12ROxAIOX-V1003 — RESI-C4-xxx ASCII Basics

Register NAME	MODBUS Register	Register VALUE	DATA TYPE	DO WRITE
ASCII COMMANDS				
HEART BEAT	ASCII READ COMMAND	#HB<CR> Result: #HB<CR>	ASCII	
	TX	#255,HB<CR>		
	RX	#255,HB<CR>		
Sends an Heartbeat to test the communication				
GET VERSION	ASCII READ COMMAND	#VERSION<CR> Result: #VERSION:<VersionHi>,<VersionMed>,<VersionLo><CR>	ASCII	
	TX	#255,VERSION<CR>		
	RX	#255,VERSION:1.1.0<CR>		
		Actual SW version:1.1.0		
Returns the version number of the module VersionHi: Version number high (1..255) VersionMed: Version number medium (1..255) VersionLo: Version number low (1..255)				
GET TYPE	ASCII READ COMMAND	#TYPE<CR> Result: #TYPE:<Type><CR>	ASCII	
	TX	#255,TYPE<CR>		
	RX	#255,TYPE:RESI-C4-A-32DI12RO16AIOX<CR>		
		Actual module type:RESI-C4-A-32DI12RO16AIOX		
Returns the actual module type				
GET FEATURES	ASCII READ COMMAND	#FTRS<CR> Result: #FTRS:<Type><CR>	ASCII	
	TX	#255,FTRS<CR>		
	RX	#255,FTRS:RESI-C4-A-32DI12RO16AIOX,RS485,DI:32,RO:12,AIOX:16<CR>		
		Actual module type:N/A		
		Number of digital inputs:N/A		
		Type of digital inputs:N/A		
Returns the actual module features				
GET OWNER	ASCII READ COMMAND	#OWNER<CR> Result: #OWNER:<Owner><CR>	ASCII	
	TX	#255,OWNER<CR>		
	RX	#255,OWNER:RESI<CR>		
		Actual owner:RESI		
Returns the actual owner of the module				

5 Dimensions of our IoT Controller

5.1 RESI-T4-xxx XT4 housing

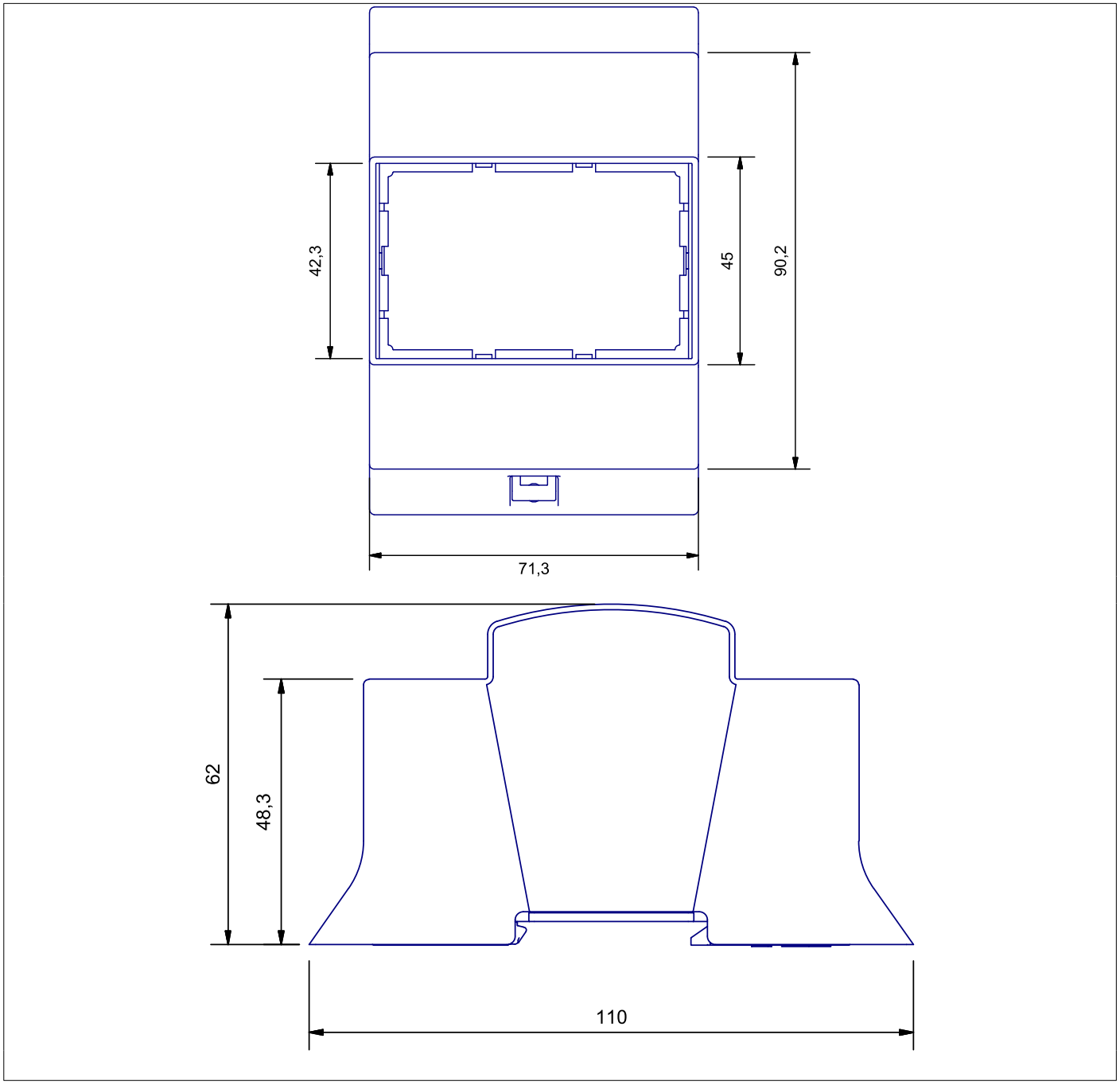


Figure: Dimensions of the housing for our T4 IoT Controller in XT4 modules in mm

Dimensions	
Housing LxWxH in mm	142.3x110x62
Color	grey RAL 7035
Material	Self-extinguishing Blend PC/ABS UL94-VO
Protection class	IP20 based on DIN 40050 / EB 60529

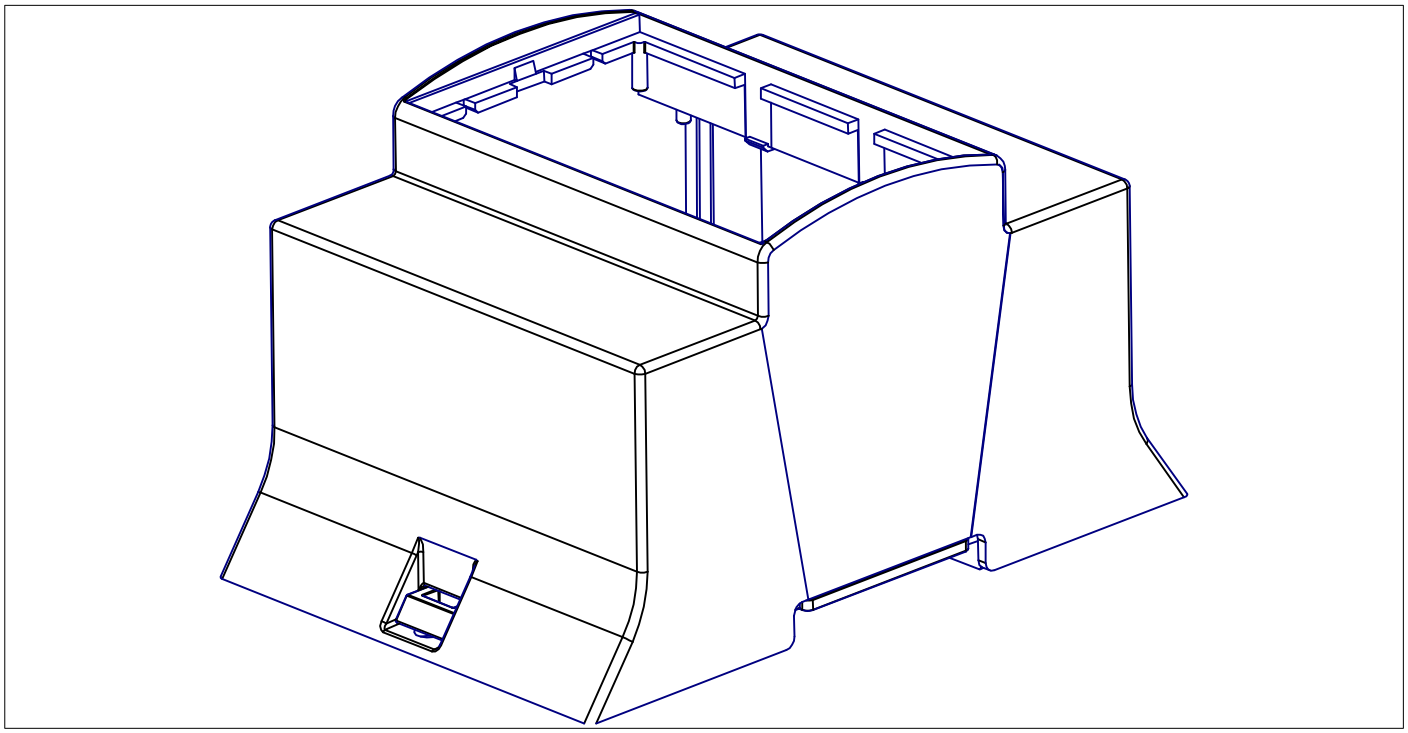


Figure: For our IoT controller in XT4 format: Housing illustration in 3D

5.2 RESI-C4-xxx XT4 housing

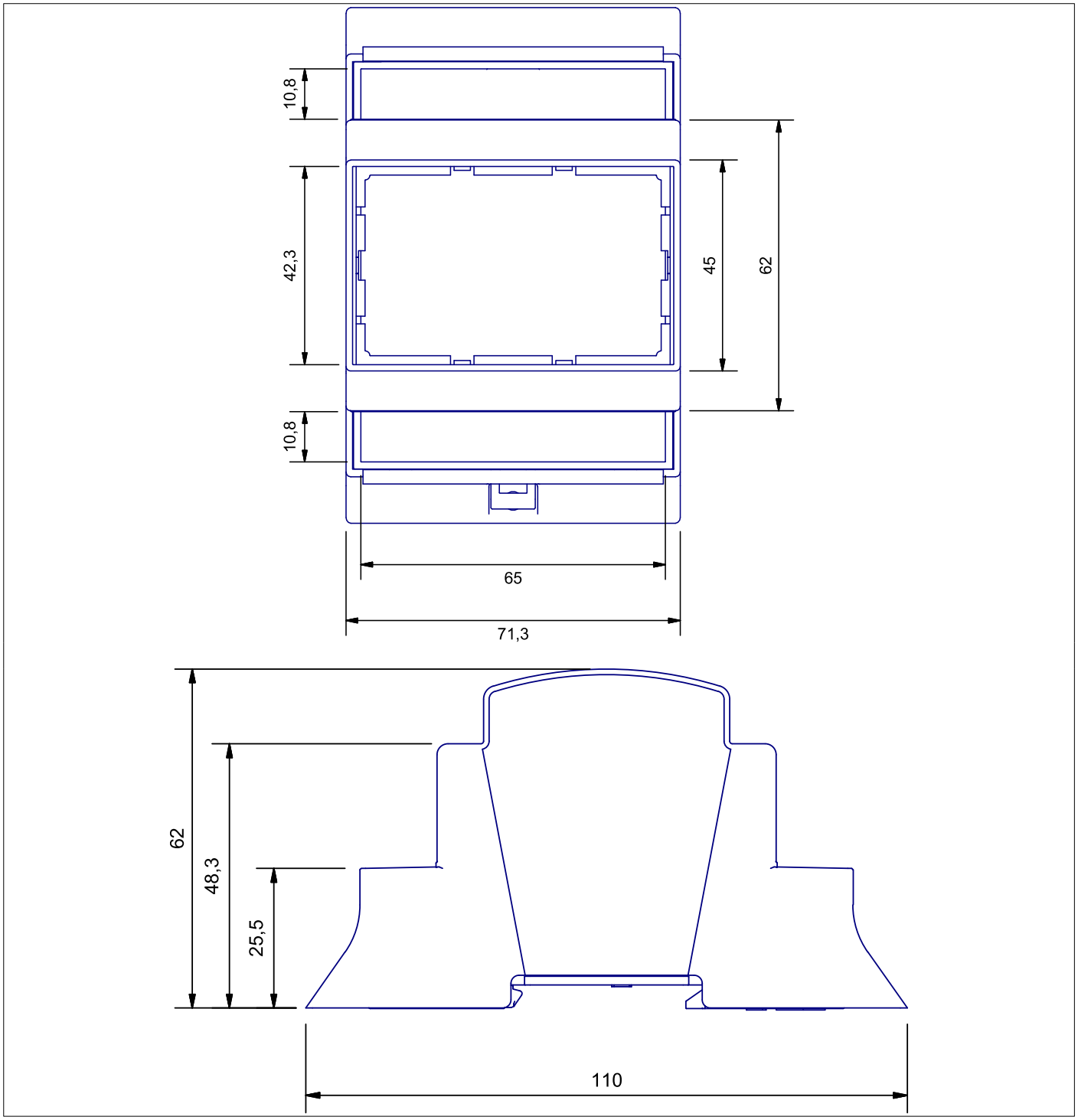


Figure: Dimensions of the housing for our C4 IoT Controller in XT4 modules in mm

Dimensions	
Housing LxWxH in mm	71.3x110x62
Color	grey RAL 7035
Material	Self-extinguishing Blend PC/ABS UL94-VO
Protection class	IP20 based on DIN 40050 / EB 60529

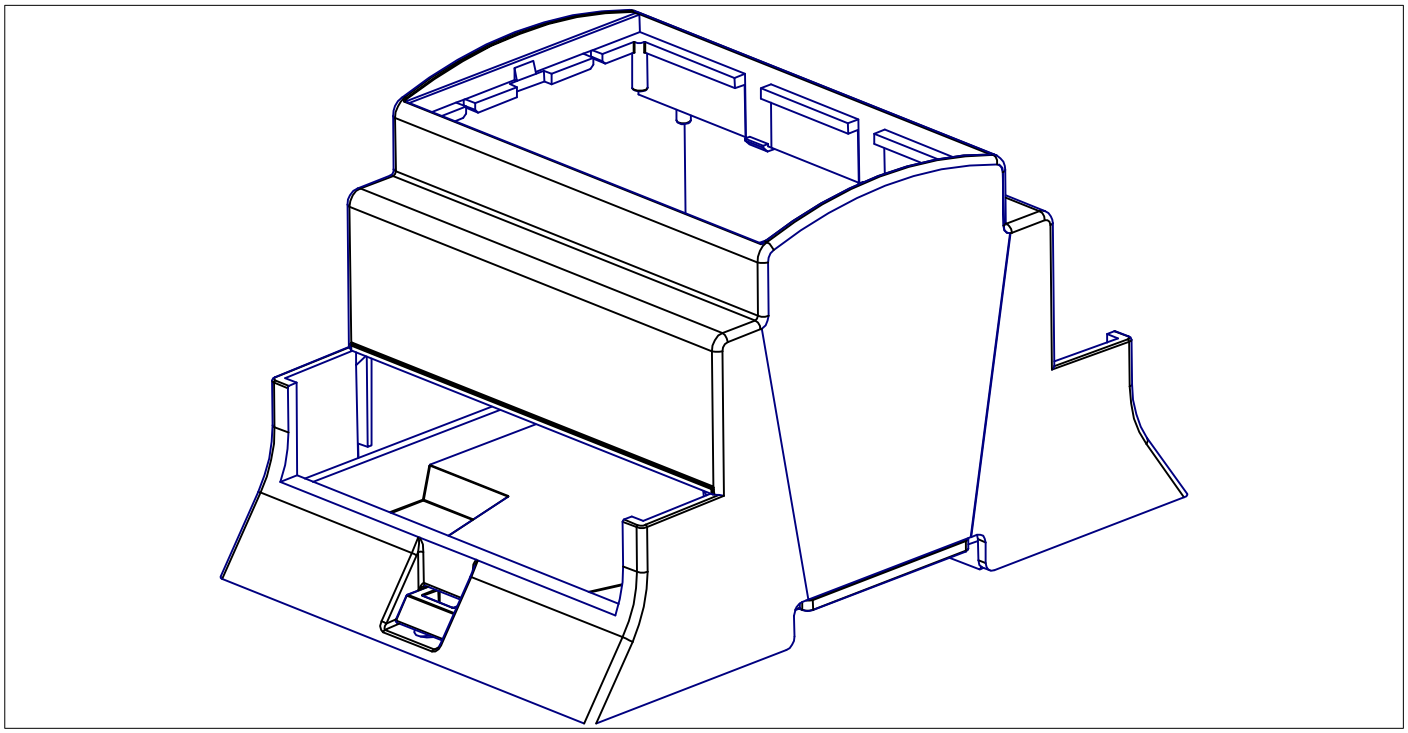


Figure: For our C4 IoT controller in XT4 format: Housing illustration in 3D

5.3 RESI-C4-xxx XT8 housing

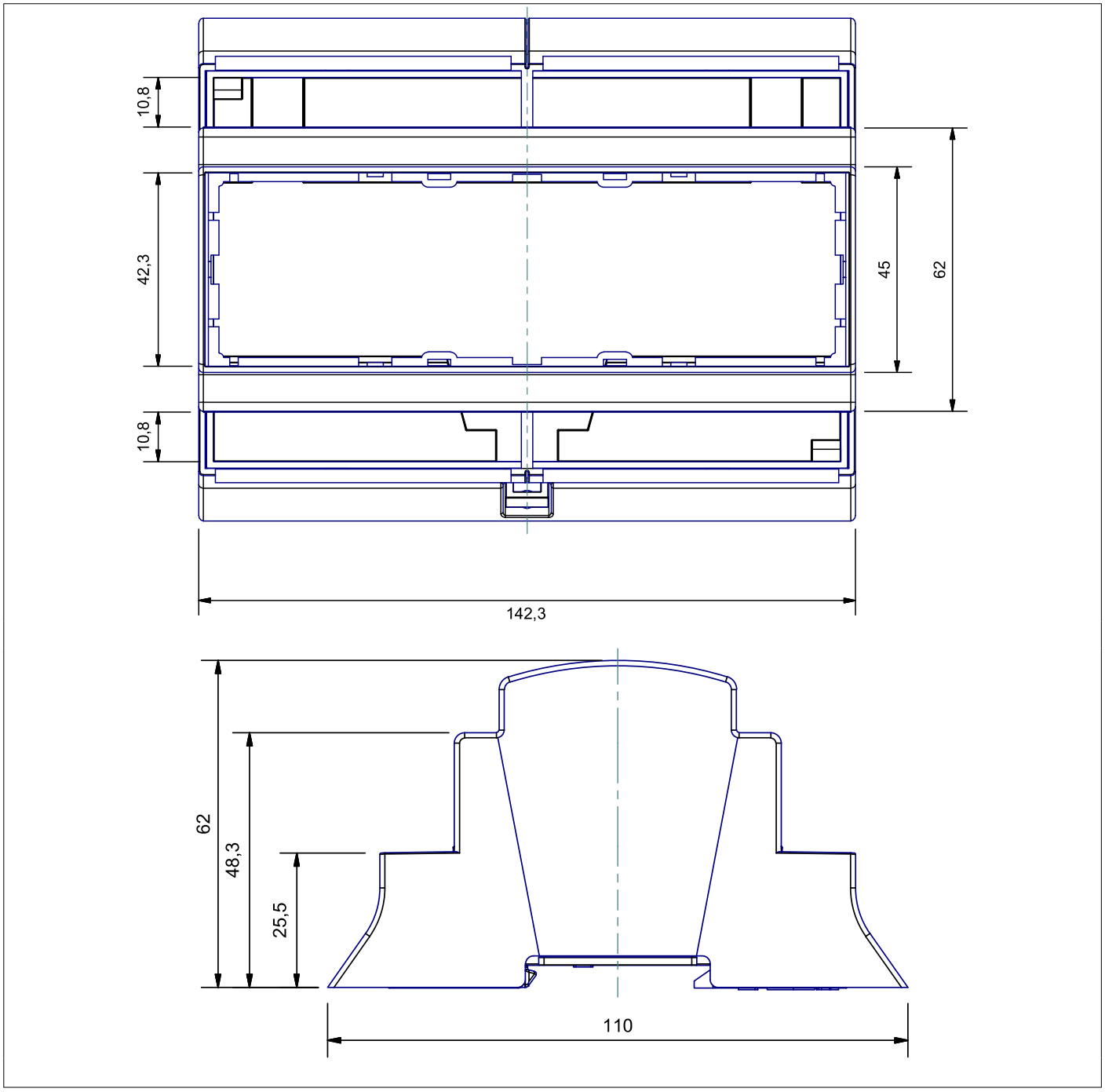


Figure: Dimensions of the housing for our C4 IoT Controller in XT8 modules in mm

Dimensions	
Housing LxWxH in mm	142.3x110x62
Color	grey RAL 7035
Material	Self-extinguishing Blend PC/ABS UL94-VO
Protection class	IP20 based on DIN 40050 / EB 60529

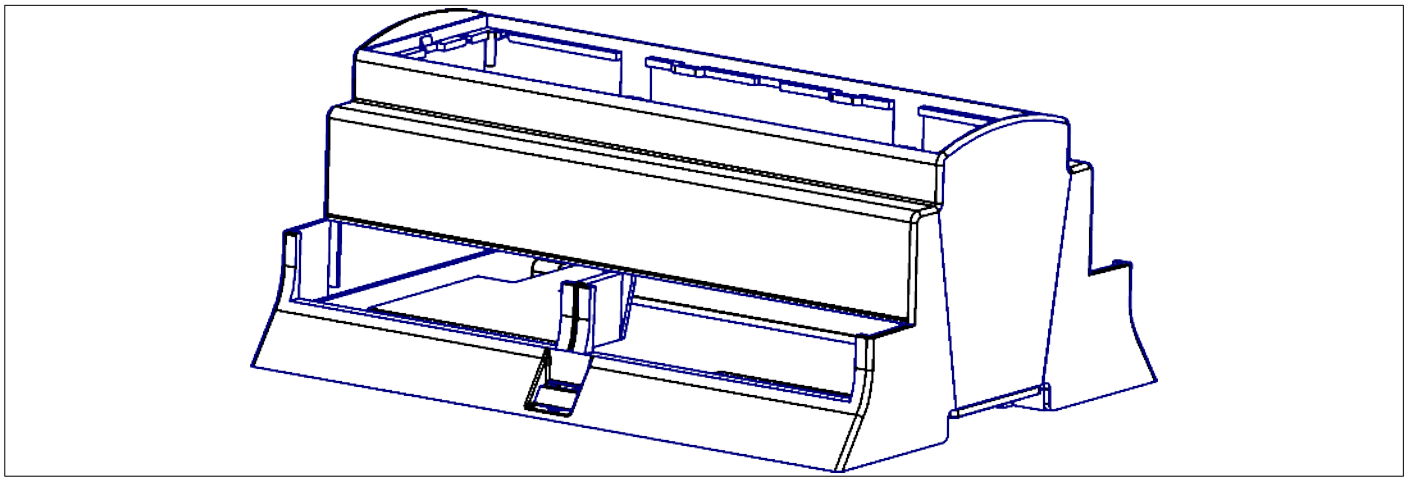


Figure: For our C4 IoT controller in XT8 format: Housing illustration in 3D

5.4 RESI-C4-xxx: XT12 housing

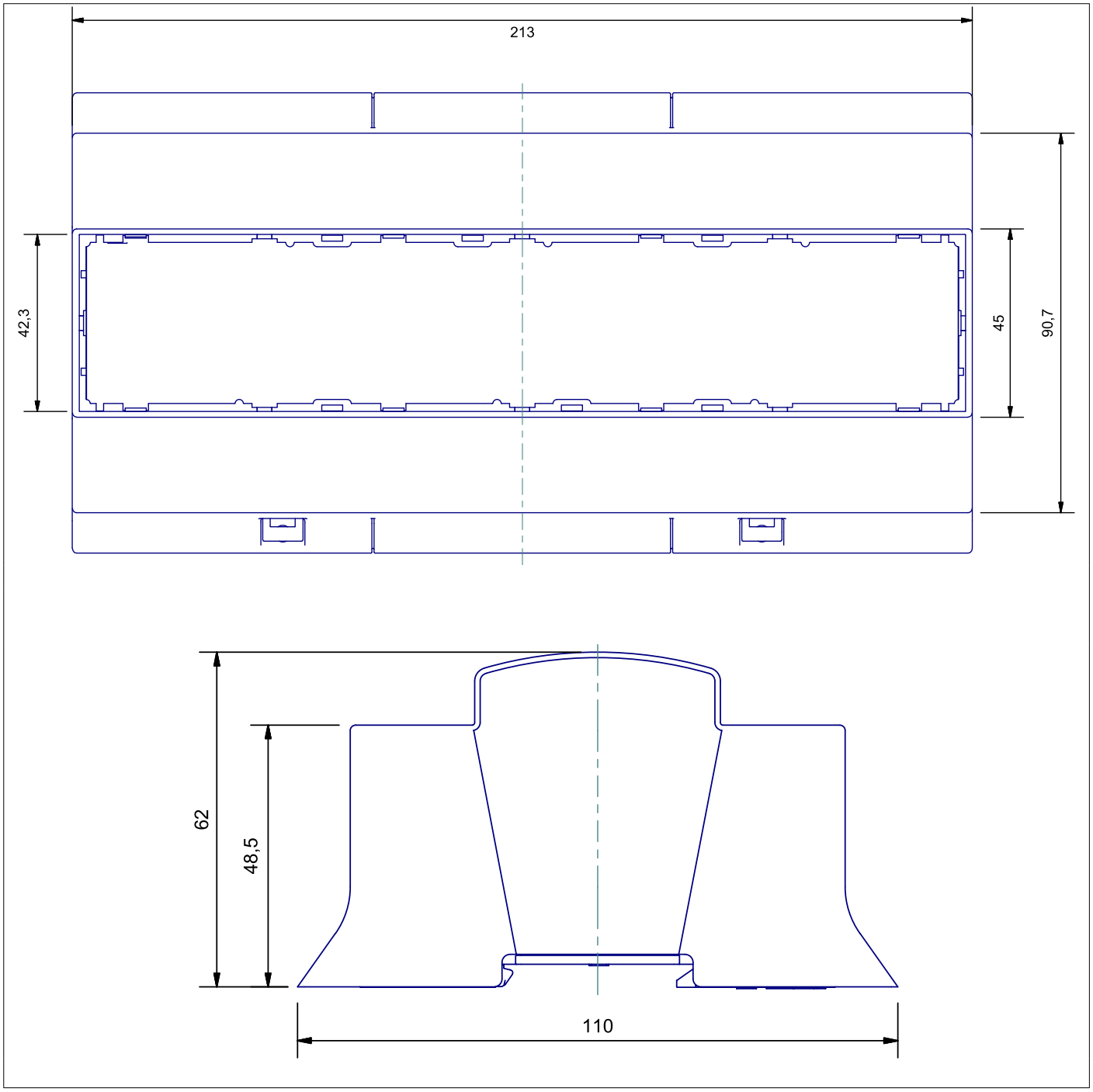


Figure: Dimensions of the housing for our C4 IoT Controller in XT12 modules in mm

Dimensions	
Housing LxWxH in mm	213x110x62
Color	grey RAL 7035
Material	Self-extinguishing Blend PC/ABS UL94-VO
Protection class	IP20 based on DIN 40050 / EB 60529

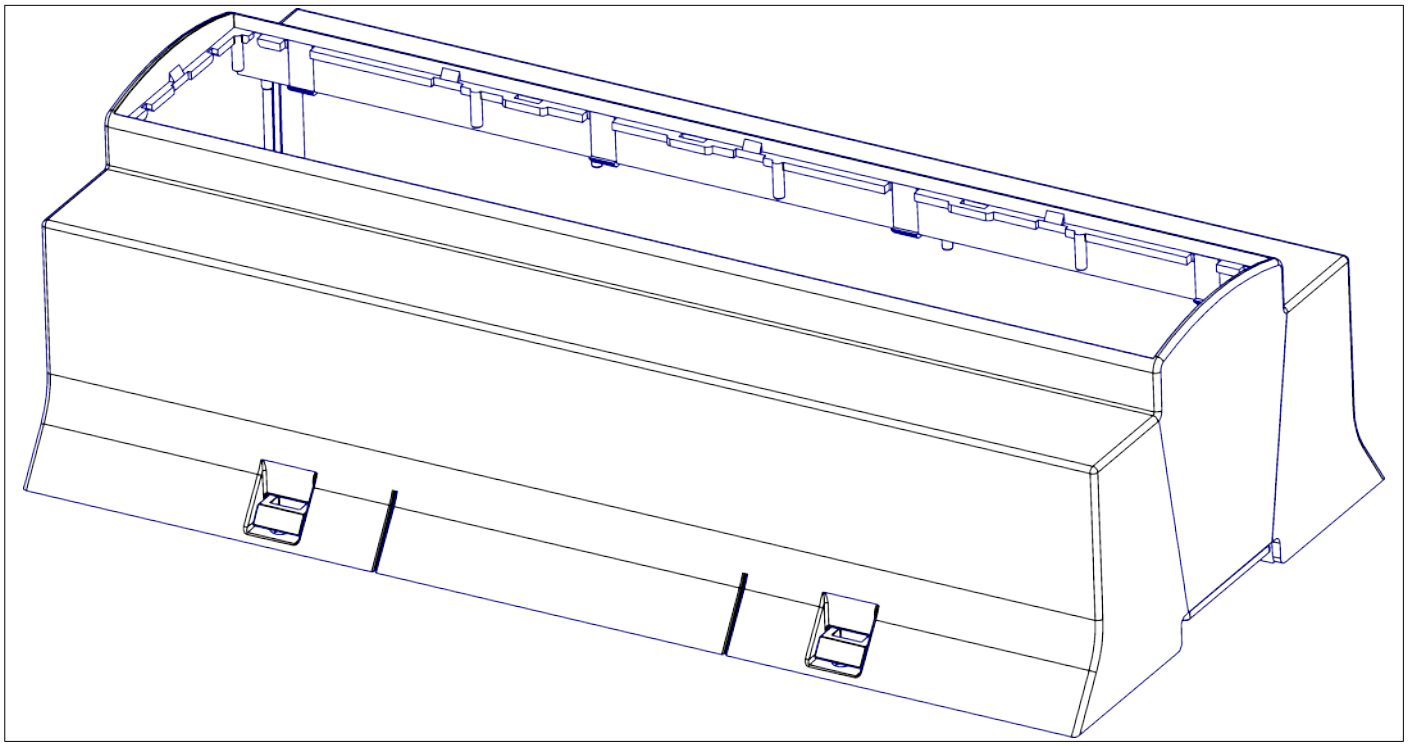


Figure: For our C4 IoT controller in XT12 format: Housing illustration in 3D

6 Common functionalities ASCII+MODBUS

This part describes the common functionality of all RESI-T4 and RESI-C4 controller.
This are the access to the LEDS and the DIP switch, the internal real time clock and the internal ferromagnetic RAM.
Also the access to all status information of an IoT controller are described here.

For every product we offer an ASCII command and MODBUS register list to know, how our co-processor can be used and how the functionality is mapped to the different coils and registers of the specific IoT controller.

6.1 Detecting the controller type and features

All of our controllers offer three basic ASCII commands:
TYPE to get the actual controller type
VERSION to read out the current software version of the ARM co-processor
FEATURES to retrieve the current features of the controller

GET VERSION	ASCII READ COMMAND	#VERSION<CR> Result: #VERSION:<VersionHi>.<VersionMed>.<VersionLo><CR>	ASCII	
	TX	#255,VERSION<CR>		
	RX	#255,VERSION:1.1.0<CR>		
		Actual SW version:1.1.0		
Returns the version number of the module VersionHi: Version number high (1..255) VersionMed: Version number medium (1..255) VersionLo: Version number low (1..255)				
GET TYPE	ASCII READ COMMAND	#TYPE<CR> Result: #TYPE:<Type><CR>	ASCII	
	TX	#255,TYPE<CR>		
	RX	#255,TYPE:RESI-C4-A-64DI60DO16AIOX<CR>		
		Actual module type:RESI-C4-A-64DI60DO16AIOX		
Returns the actual module type				
GET FEATURES	ASCII READ COMMAND	#FTRS<CR> Result: #FTRS:<Type><CR>	ASCII	
	TX	#255,FTRS<CR>		
	RX	#255,FTRS:RESI-C4-A-64DI60DO16AIOX,RS485,DI:64,DO:60,AIOX:16,DOIC:MAX14915<CR>		
		Actual module type:N/A		
		Number of digital inputs:N/A		
		Type of digital inputs:N/A		
Returns the actual module features				

The same information can be retrieved from the MODBUS registers:

RESI Configurator RESI-C4-64DI60DO,64DI60DOxAIOX-V1003 — RESI-C4-xxx MB Basics

Register NAME	MODBUS Register	Register VALUE	NEW REAL VALUE	NEW VALUE	DATA TYPE	DO WRITE
PRODUCT INFO						
HW_GROUP	3x65201 4x65201 I:65200	50176,0xC400 B:C4 00			UINT16 R/O	
This is the group of hardware of the current product						
HW_TYPE	3x65202 4x65202 I:65201	1,0x0001 B:00 01			UINT16 R/O	
This is the type of hardware of the current product						
SW_VERSION	3x65203 4x65203 I:65202	272,0x0110 B:01 10			UINT16 R/O	
		SW VERSION:0.1.0				
This is the current software version of the firmware						
SW_AUTHOR	3x65204 4x65204 I:65203	21321,0x5349 B:53 49			UINT16 R/O	
This is the current software author of the firmware						
MANUFACTURER	3x65205 4x65205 I:65204	1380275017,0x52455349 B:52 45 53 49			UINT32 R/O	
This is the current software author of the firmware						
NUMBER OF DIGITAL INPUTS	3x65207 4x65207 I:65206	0,0x0000 B:00 00			UINT16 R/O	
		Number of DIS:0				
This is the current software version of the firmware						
NUMBER OF DIGITAL OUTPUTS	3x65208 4x65208 I:65207	0,0x0000 B:00 00			UINT16 R/O	
		Number of DOS:0				
This is the current software version of the firmware						
NUMBER OF ANALOG INPUTS	3x65209 4x65209 I:65208	0,0x0000 B:00 00			UINT16 R/O	
		Number of AIS:0				
This is the current software version of the firmware						
NUMBER OF ANALOG OUTPUTS	3x65210 4x65210 I:65209	0,0x0000 B:00 00			UINT16 R/O	
		Number of AOS:0				
This is the current software version of the firmware						
NUMBER OF UNIVERSAL IN/OUTPUTS	3x65211 4x65211 I:65210	0,0x0000 B:00 00			UINT16 R/O	
		Number of AIOX:0				
This is the current software version of the firmware						
NUMBER OF SPECIAL INPUTS	3x65212 4x65212 I:65211	0,0x0000 B:00 00			UINT16 R/O	
		Number of special inputs:0				
This is the current software version of the firmware						
NUMBER OF SPECIAL OUTPUTS	3x65213 4x65213 I:65212	0,0x0000 B:00 00			UINT16 R/O	
		Number of special outputs:0				
This is the current software version of the firmware						
FEATURE1	3x65214 4x65214 I:65213	2,0x0002 B:00 02			UINT16 R/O	
		Feature:RS485				
This is the feature list of the controller: 0:NONE, 1:RS232, 2:RS485, 3:KNX, 4:DALI, 5:MBUS, 6:LORA, 7:LTE, 8:2xETHERNET						
FEATURE2	3x65215 4x65215 I:65214	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				
FEATURE3	3x65216 4x65216 I:65215	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				
FEATURE4	3x65217 4x65217 I:65216	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				
FEATURE5	3x65218 4x65218 I:65217	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				
FEATURE6	3x65219 4x65219 I:65218	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				
FEATURE7	3x65220 4x65220 I:65219	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				

FEATURE8	3x65221 4x65221 I:65220	0,0x0000 B:00 00			UINT16 R/O	
		Feature:NONE				

6.2 Using the LEDs and DIP switch

Most of our IoT controller offer four LEDs organized in three LED outlets in the cover plate.

- LED1: GREEN: DATA LED
- LED2: WHITE: STATE LED and LED3: RED ERROR LED
- LED4: YELLOW: INFO LED

Except of the INFO LED, which is used in some IoT controllers internally you can use this LEDs in your software to signal special states.

Also most modules offer an 8-pin DIP switch. Again this DIP switch is only for your software.

6.2.1 Reading the DIP switch in ASCII+MODBUS

To read the DIP switch you have to use the ASCII command GDIP. Below you see the specification in our document. The co-processor returns the current status for the DIP switch as described below.

GET DIP SWITCH	ASCII READ COMMAND	#GDIP<CR> Result: #GDIP:<DIPSwitchDec>,<DIPSwitchHex><CR>	ASCII	
	TX	#255,GDIP<CR>		
	RX	#255,GDIP:0,0x0<CR>		
		Actual DIP SWITCH settings:00000000		
Returns the actual setting of the Dip switches as decimal number and as hexadecimal number. DIPSwitchDec DIPSwitchHex The current value of the DIP switches: Bit 0: DIP Switch 1 (=0:OFF, =1:ON) Bit 1: DIP Switch 2 (=0:OFF, =1:ON) Bit 2: DIP Switch 3 (=0:OFF, =1:ON) Bit 3: DIP Switch 4 (=0:OFF, =1:ON) Bit 4: DIP Switch 5 (=0:OFF, =1:ON) Bit 5: DIP Switch 6 (=0:OFF, =1:ON) Bit 6: DIP Switch 7(=0:OFF, =1:ON) Bit 7: DIP Switch 8 (=0:OFF, =1:ON)				

You can also use MODBUS registers and coils get the current status of the DIP switch settings too. Search the PDF document for DIP to find all possible MODBUS registers and coils:

DIP SWITCH STATUS						
DIP SWITCH	3x65501 4x65501 I:65500	85,0x0055 B:00 55			UINT16 R/O	
Returns the actual setting of the Dip switches. Bit 0: DIP Switch 1 (=0:OFF, =1:ON) Bit 1: DIP Switch 2 (=0:OFF, =1:ON) Bit 2: DIP Switch 3 (=0:OFF, =1:ON) Bit 3: DIP Switch 4 (=0:OFF, =1:ON) Bit 4: DIP Switch 5 (=0:OFF, =1:ON) Bit 5: DIP Switch 6 (=0:OFF, =1:ON) Bit 6: DIP Switch 7 (=0:OFF, =1:ON) Bit 7: DIP Switch 8 (=0:OFF, =1:ON)						

Register NAME	MODBUS Register	Register VALUE	NEW REAL VALUE	NEW VALUE	DATA TYPE	DO WRITE
DIP SWITCH STATUS						
DIP SWITCH DIP1	1x65001 2x65001 I:65000	????			BIT R/O	
Returns the actual setting of the Dip switches. =0: DIP is OFF =1: DIP is ON						
DIP SWITCH DIP2	1x65002 2x65002 I:65001	????			BIT R/O	
DIP SWITCH DIP3	1x65003 2x65003 I:65002	????			BIT R/O	
DIP SWITCH DIP4	1x65004 2x65004 I:65003	????			BIT R/O	
DIP SWITCH DIP5	1x65005 2x65005 I:65004	????			BIT R/O	
DIP SWITCH DIP6	1x65006 2x65006 I:65005	????			BIT R/O	
DIP SWITCH DIP7	1x65007 2x65007 I:65006	????			BIT R/O	
DIP SWITCH DIP8	1x65008 2x65008 I:65007	????			BIT R/O	

6.2.2 Update the LEDs in ASCII+MODBUS

You can switch every LED to ON, OFF. You can INVERT the current LED state. And you can blink and flash the LED with different timings. Also you can create a one time pulse on the LED. Additional commands retrieve the current LED status.

In ASCII you can use the following commands for all four LEDs:

- LED1: GREEN: DATA LED
- LED2: WHITE: STATE LED
- LED3: RED ERROR LED
- LED4: YELLOW: INFO LED

Register NAME	MODBUS Register	Register VALUE	DATA TYPE	DO WRITE
LED STATUS:LED1:GREEN				
GET LED1	ASCII READ COMMAND	#GLED1<CR> Result: #GLED1:<LEDMode>,<LEDStateDec>,<LEDStateHex><CR>	ASCII	
	TX	#255,GLED1<CR>		
	RX	#255,GLED1:OFF,0.0x0<CR>		
		Actual LED state:OFF LED ist currently 0		
Returns the actual state of the LED1:GREEN on the cover of module				
LED COMMANDS:LED1:GREEN				
SET LED1 OFF	ASCII WRITE COMMAND	#SL1OFF<CR> Result: #OK<CR>	ASCII	YES
	TX	#255,SL1OFF<CR>		
	RX	#255,OK<CR>		
Sets the current state of the LED1:GREEN on the cover of module to OFF				
SET LED1 ON	ASCII WRITE COMMAND	#SL1ON<CR> Result: #OK<CR>	ASCII	YES
	TX	#255,SL1ON<CR>		
	RX	#255,OK<CR>		
Sets the current state of the LED1:GREEN on the cover of module to ON				
SET LED1 INVERT	ASCII WRITE COMMAND	#SL1INV<CR> Result: #OK<CR>	ASCII	YES
	TX	#255,SL1INV<CR>		
	RX	#255,OK<CR>		
Inverts the current state of the LED1:GREEN on the cover of module from ON to OFF or from OFF to ON				
SET LED1 PULSE	ASCII WRITE COMMAND	#SL1PULSE:<PULSETIME><CR> Result: #OK<CR>	ASCII	YES
	PULSETIME	1000		
	TX	#255,SL1PULSE:1000<CR>		
	RX	#255,OK<CR>		
Sets the current state of the LED1:GREEN on the cover of module to PULSE and defines the one time pulse duration in Milliseconds between 1 and 60000				
SET LED1 BLINK	ASCII WRITE COMMAND	#SL1BLINK:<BLINKTIME><CR> Result: #OK<CR>	ASCII	YES
	BLINKTIME	1000		
	TX	#255,SL1BLINK:1000<CR>		
	RX	#255,OK<CR>		
Sets the current state of the LED1:GREEN on the cover of module to PULSE and defines the one time pulse duration in Milliseconds between 1 and 60000				
SET LED1 FLASH	ASCII WRITE COMMAND	#SL1FLASH:<ONTIME>,<OFFTIME><CR> Result: #OK<CR>	ASCII	YES
	ONTIME	200		
	OFFTIME	3000		
	TX	#255,SL1FLASH:200,3000<CR>		
	RX	#255,OK<CR>		
Sets the current state of the LED1:GREEN on the cover of module to FLASH and defines the on and off intervals in Milliseconds between 20 and 600000				

On the MODBUS you can use coils to define the LED functionality:

LED1:GREEN						
LED1:GREEN SET TO OFF	1x65009 2x65009 I:65008	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil sets the LED to OFF						
LED1:GREEN SET TO ON	1x65010 2x65010 I:65009	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil sets the LED to ON						
LED1:GREEN INVERT LED STATE	1x65011 2x65011 I:65010	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil inverts the actual LED state						
LED1:GREEN BLINK	1x65012 2x65012 I:65011	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start symmetrical blinking of LED with last defined time						
LED1:GREEN FLASH	1x65013 2x65013 I:65012	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start asymmetrical flashing of LED with last defined times						
LED1:GREEN PULSE	1x65014 2x65014 I:65013	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start one time pulse of LED with last defined time						
LED1:GREEN BLINK 5s	1x65015 2x65015 I:65014	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start symmetrical blinking of LED with 5s ON-5s OFF cycle						
LED1:GREEN BLINK 1s	1x65016 2x65016 I:65015	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start symmetrical blinking of LED with 1s ON-1s OFF cycle						
LED1:GREEN BLINK 250ms	1x65017 2x65017 I:65016	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start symmetrical blinking of LED with 250ms ON-250ms OFF cycle						
LED1:GREEN BLINK 50ms	1x65018 2x65018 I:65017	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start symmetrical blinking of LED with 50ms ON-50ms OFF cycle						
LED1:GREEN FLASH 5s-1s	1x65019 2x65019 I:65018	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start asymmetrical flashing of LED with 5s ON-1s OFF cycle						
LED1:GREEN FLASH 1s-250ms	1x65020 2x65020 I:65019	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start asymmetrical flashing of LED with 1s ON-250ms OFF cycle						
LED1:GREEN FLASH 500ms-100ms	1x65021 2x65021 I:65020	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start asymmetrical flashing of LED with 500ms ON-100ms OFF cycle						
LED1:GREEN FLASH 300ms-50ms	1x65022 2x65022 I:65021	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start asymmetrical flashing of LED with 300ms ON-50ms OFF cycle						
LED1:GREEN PULSE 1s	1x65023 2x65023 I:65022	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start one time pulse of LED with 1s ON						
LED1:GREEN PULSE 500ms	1x65024 2x65024 I:65023	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start one time pulse of LED with 500ms ON						
LED1:GREEN PULSE 250ms	1x65025 2x65025 I:65024	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start one time pulse of LED with 250ms ON						
LED1:GREEN PULSE 100ms	1x65026 2x65026 I:65025	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start one time pulse of LED with 100ms ON						
LED1:GREEN PULSE 20ms	1x65027 2x65027 I:65026	????		N/A:DO NOTHING	BIT W/O	NO
Writing 1 to this coil start one time pulse of LED with 20ms ON						

Or you use registers:

LED2:WHITE						
LED2:WHITE STATE	3x65505 4x65505 I:65504	????		1:SET TO ON	UINT16 R/W	NO
		State of LED:????				
Returns the actual state of the LED Writing to this register will set a new state for the LED 0: Switch LED permanent OFF 1: Switch LED permanent ON 2: Invert last state of LED 3: Start symmetrical blinking of LED with TIME1 ON and TIME1 OFF 4: Start asymmetrical flashing of LED with TIME1 ON and TIME2 OFF 5: Start one time pulse of LED with TIME1 ON and inifinite OFF						
LED2:WHITE TIME1	3x65506 4x65506 I:65505	????		1000	UINT16 R/W	YES
		Actual time 1 in ms:0				
Returns the actual time1 for blink,flash and pulse ON time in Milliseconds Writing to this register sets a new time in the range 20-65534ms						
LED2:WHITE TIME2	3x65507 4x65507 I:65506	????		2000	UINT16 R/W	YES
		Actual time 2 in ms:0				
Returns the actual time2 for blink and flash OFF time in Milliseconds Writing to this register sets a new time in the range 20-65534ms						

6.2.3 Use the real-time clock

Our ARM co-processor offers an internal real-time clock with external capacitor backup if power fails.
Therefore you can use this RTC for your internal purposes as your date & time source. If you want to synchronize this RTC with the LINUX date & time, you have to write code for this by yourself. This RTC is completely independent from the LINUX.

First of all you can check the current capacitor voltage with the command GCPUBACK. This is the voltage of the external capacitor, which will be loaded during power-on of the controller and used to drive the RTC while power is off. It should be over 3V.

CPU PARAMETERS						
GET CPU VOLTAGE	ASCII READ COMMAND	#GCPUTEMP<CR> Result: #GCPUTEMP:<CPUTemp><CR>			ASCII	
	TX	#255,GCPUTEMP<CR>				
	RX	#255,GCPUTEMP:41.4092<CR>				
		Actual internal temperature of CPU:41.4092°C				
Current internal temperature of CPU in ° Celsius.						
GET CPU VOLTAGE	ASCII READ COMMAND	#GCPUVOLT<CR> Result: #GCPUVOLT:<CPUVoltage><CR>			ASCII	
	TX	#255,GCPUVOLT<CR>				
	RX	#255,GCPUVOLT:3.3453<CR>				
		Actual supply voltage of CPU:3,3453V				
Current internal supply voltage of CPU in Volt.						
GET CPU BACKUP	ASCII READ COMMAND	#GCPUBACK<CR> Result: #GCPUBACK:<CPUBackupVoltage><CR>			ASCII	
	TX	#255,GCPUBACK<CR>				
	RX	#255,GCPUBACK:3.1871<CR>				
		Actual backup voltage of CPU for RTC:3.1871V				
Current internal backup voltage of CPU for the RTC in Volt.						

Or with MODBUS registers:

CPU TEMPERATURE	3x65527 4x65527 I:65526	5061,0x13C5 B:13 C5			UINT16 R/O	
		Actual internal temperature of CPU:50,61°C				
Current internal temperature of CPU in ° Celsius multiplied by 100.						
CPU VOLTAGE	3x65528 4x65528 I:65527	333,0x014D B:01 4D			UINT16 R/O	
		Actual supply voltage of CPU:3,33V				
Current internal supply voltage of CPU in Volt multiplied by 1000.						
CPU BACKUP	3x65529 4x65529 I:65528	311,0x0137 B:01 37			UINT16 R/O	
		Actual backup voltage of CPU for RTC:3,11V				
Current internal backup voltage of CPU for RTC in Volt multiplied by 1000.						

Then you can read or write the current RTC date and time with:

ASCII COMMANDS:REAL TIME CLOCK				
GET REAL TIME CLOCK	ASCII READ COMMAND	#GRTC<CR> Result: #GRTC:YMD,<YEAR>,<MONTH>,<DAY>,HMS,<HOUR>,<MINUTE>,<SECOND>,<WEEKDAY> ,DOK,<DATEOK>,TOK,<TIMEOK><CR>	ASCII	
	TX	#255,GRTC<CR>		
	RX	#255,GRTC:YMD,24,1,5,HMS,15,34,47,FRI,DOK,1,TOK,1<CR>		
		Actual date DD.MM.YYYY:5.1.2024		
		Actual time HH.MM.SS (24h):15:34:47		
		Actual Weekday:FRI		
		Battery buffered date is ok:YES		
		Battery buffered time is ok:YES		
Shows current RTC time of battery backup RTC on module				
ASCII COMMANDS:REAL TIME CLOCK				
SET REAL TIME CLOCK	ASCII WRITE COMMAND	#SRTC:YMD,<YEAR>,<MONTH>,<DAY>,HMS,<HOUR>,<MINUTE>,<SECOND>, <WEEKDAY><CR> Result: #OK<CR>	ASCII	YES
	YEAR	2024		
	MONTH	03		
	DAY	09		
	HOUR	10		
	MINUTE	42		
	SECOND	43		
	WEEKDAY	SAT		
	TX	#255,SRTC:YMD,24,03,09,HMS,10,42,43,SAT<CR>		
	RX	#255,OK<CR>		
Executes a software reset (Reboot) of the module.				

Again with MODBUS registers:

RTC REAL TIME CLOCK						
RTC YEAR	3x65231 4x65231 I:65230	24,0x0018 B:00 18		24	UINT16 R/W	NO
		Actual RTC year:24				
Returns the actual year of the internal real time clock in the range of 24 to 99. Writing to this register prepares the setting of a new time.						
RTC MONTH	3x65232 4x65232 I:65231	2,0x0002 B:00 02		1	UINT16 R/W	NO
		Actual RTC month:2				
Returns the actual month of the internal real time clock in the range of 1 to 12 Writing to this register prepares the setting of a new time.						
RTC DAY	3x65233 4x65233 I:65232	29,0x001D B:00 1D		1	UINT16 R/W	NO
		Actual RTC day:29				
Returns the actual day of the internal real time clock in the range of 1 to 31 Writing to this register prepares the setting of a new time.						
RTC HOUR	3x65234 4x65234 I:65233	18,0x0012 B:00 12		12	UINT16 R/W	NO
		Actual RTC month:18				
Returns the actual hour of the internal real time clock in the range of 0 to 23 Writing to this register prepares the setting of a new time.						
RTC MINUTE	3x65235 4x65235 I:65234	0,0x0000 B:00 00		45	UINT16 R/W	NO
		Actual RTC hour:0				
Returns the actual minute of the internal real time clock in the range of 0 to 59 Writing to this register prepares the setting of a new time.						
RTC SECOND	3x65236 4x65236 I:65235	23,0x0017 B:00 17		30	UINT16 R/W	NO
		Actual RTC second:23				
Returns the actual second of the internal real time clock in the range of 0 to 59 Writing to this register prepares the setting of a new time.						
RTC DAY OF WEEK	3x65237 4x65237 I:65236	4,0x0004 B:00 04		5:FRIDAY	UINT16 R/W	NO
		Actual RTC week day:THU		SELECT DAY OF WEEK		
Returns the actual day of week in the range 1 to 7 1:MON, 2:TUE, 3:WED, 4:THU, 5:FRI, 6:SAT, 7:SUN Writing to this register writes a new date and time and weekday to the RTC						

6.2.4 Retrieve the unique serial number+box name

Our ARM Co-processor offers a unique 96 bit serial number for your software licencing. Together with the parameter BOXNAME you can use this to protect your software running only on a specific device:

Use the following ASCII commands to set an individual BOX name:

SET BOX NAME	ASCII WRITE COMMAND	#SETBOXNAME:<BOXNAME><CR> Result: #OK<CR>	ASCII	YES
	BOXNAME	MYBOX		
	TX	#255,SETBOXNAME:MYBOX<CR>		
	RX	#255,OK<CR>		
Sets a new box name for the controller				
GET BOX NAME	ASCII READ COMMAND	#BOXNAME<CR> Result: #BOXNAME:<BoxName><CR>	ASCII	
	TX	#255,BOXNAME<CR>		
	RX	#255,BOXNAME:MYBOX<CR>		
		Actual box name:MYBOX		
Returns the actual box name of the module. If no box name is defined, the value NONAME is returned				

And retrieve the unique serial number with:

GET SERIAL NUMBER	ASCII READ COMMAND	#SN<CR> Result: #SN:<Serial><CR>	ASCII	
	TX	#255,SN<CR>		
	RX	#255,SN:39002C000253554637303820<CR>		
		Actual serial number:39002C000253554637303820		
Returns the actual serial number of the module				

The BOX name is not accessible via MODBUS, but the serial number can be read with:

CPU DATA						
SERIAL1	3x65521 4x65521 I:65520	34,0x0022 B:00 22			UINT16 R/O	
Serial number of module as 96 bit unsigned integer number						
SERIAL2	3x65522 4x65522 I:65521	24,0x0018 B:00 18			UINT16 R/O	
SERIAL3	3x65523 4x65523 I:65522	22291,0x5713 B:57 13			UINT16 R/O	
SERIAL4	3x65524 4x65524 I:65523	20547,0x5043 B:50 43			UINT16 R/O	
SERIAL5	3x65525 4x65525 I:65524	13361,0x3431 B:34 31			UINT16 R/O	
SERIAL6	3x65526 4x65526 I:65525	8246,0x2036 B:20 36			UINT16 R/O	
		SERIAL:22001800135743503134362O				
Serial number of module as 96 bit unsigned integer number						

6.2.5 Use the ferromagnetic RAM

Our ARM Co-processor has a build in ferromagnetic RAM (FRAM) of 2kB. The read or write access cycles to this FRAM is almost unlimited! All values are stored permanently. So after a power on all previous written values are restored. You can rewrite every byte without the need to delete a bank or sector like a EEPROM or FLASH.

The controller uses this FRAM for storing all IO watchdog values for digital and analog outputs and the IO watchdog timing. So you cannot use all of the 2kB for your application.

With the ASCII command GFRAMSIZE you can read the total size of the FRAM and the internal used bytes e.g. 210 bytes:

ASCII COMMANDS:FRAM			
GET FRAMSIZE	ASCII READ COMMAND	#GFRAMSIZE<CR> Result: #GFRAMSIZE:<FRAMType>,<FRAMSize>,<UsedSizeDEC>,<UsedSizeHEX><CR>	ASCII
	TX	#255,GFRAMSIZE<CR>	
	RX	#255,GFRAMSIZE:FM25L16B_G,2kB,210,0xD2<CR>	
Reads the actual type and size of the used FRAM. The <UsedSize> describes the internal used space in bytes of the FRAM			

With the ASCII commands GFRAM16, GFRAM32, GFRAMDBL you can read every FRAM memory location. Be aware that the index 0 is the first byte you can use, you cannot write or read the internal used FRAM memory area.
Also you have to think about the byte size of your data, FRAM16 uses 2 bytes, FRAM32 4 bytes and FRAMDBL uses 8 bytes.
Writing to this memory will be done with SFRAM16, SFRAM32 or SFRAMDBL. You can use every byte index as a start. But it makes sense to be even aligned!
BE AWARE: Writing a 32 bit value starting at byte 10 and the writing a 16 bit value starting at 12, will overwrite half of the previous stored 32 bit value!

So if you write e.g. four 16 bit values you have to use the index 0, 2, 4 and 6
Writing e.g. 3 double values you have to use index 0, 8 and 16

GET FRAM16	ASCII READ COMMAND	#GFRAM16:<INDEX><CR> Result: #GFRAM16:<INDEXDEC>,<VALUEDEC>,<INDEXHEX>,<VALUEHEX><CR> or #GFRAM16:<INDEXDEC>,ERR,<INDEXHEX>,ERR<CR>	ASCII	
	INDEX	350		
	TX	#255,GFRAM16:350<CR>		
	RX	#255,GFRAM16:350,0,0x15E,0x0<CR>		
		FRAM Index in bytes:350		
		FRAM Value in decimal:0		
Reads the actual UINT16 value (2 bytes) of FRAM memory <INDEX>. <INDEX> is a BYTE index in the FRAM strogae starting with 0.				
GET FRAM32	ASCII READ COMMAND	#GFRAM32:<INDEX><CR> Result: #GFRAM32:<INDEXDEC>,<VALUEDEC>,<INDEXHEX>,<VALUEHEX><CR> or #GFRAM32:<INDEXDEC>,ERR,<INDEXHEX>,ERR<CR>	ASCII	
	INDEX	350		
	TX	#255,GFRAM32:350<CR>		
	RX	#255,GFRAM32:350,0,0x15E,0x0<CR>		
		FRAM Index in bytes:350		
		FRAM Value in decimal:0		
Reads the actual UINT32 value 4 bytes) of FRAM memory <INDEX>. <INDEX> is a BYTE index in the FRAM strogae starting with 0.				
GET FRAMDBL	ASCII READ COMMAND	#GFRAMDBL:<INDEX><CR> Result: #GFRAMDBL:<INDEXDEC>,<VALUEDBL>,<INDEXHEX>,<VALUEDBL><CR> or #GFRAMDBL:<INDEXDEC>,ERR,<INDEXHEX>,ERR<CR>	ASCII	
	INDEX	400		
	TX	#255,GFRAMDBL:400<CR>		
	RX	#255,GFRAMDBL:400,0,0x190,0<CR>		
		FRAM Index in bytes:400		
		FRAM Value in decimal:0		
Reads the actual DOUBLE value 8 bytes) of FRAM memory <INDEX>. <INDEX> is a BYTE index in the FRAM strogae starting with 0.				

SET FRAM16	ASCII WRITE COMMAND	#SFRAM16:<INDEX>,<VALUE><CR> Result: #SFRAM16:OK<CR> or #SFRAM16:ERR<CR>	ASCII	YES
	INDEX	350		
	VALUE	1234		
	TX	#255,SFRAM16:350,1234<CR>		
	RX	#255,SFRAM16:OK<CR>		
Writes a new UINT16 value (2 byte) into FRAM memory <INDEX>. <INDEX> is a BYTE index in the FRAM strogae starting with 0.				
SET FRAM32	ASCII WRITE COMMAND	#SFRAM32:<INDEX>,<VALUE><CR> Result: #SFRAM32:OK<CR> or #SFRAM32:ERR<CR>	ASCII	YES
	INDEX	350		
	VALUE	123456		
	TX	#255,SFRAM32:350,123456<CR>		
	RX	#255,SFRAM32:OK<CR>		
Writes a new UINT32 value (4 byte) into FRAM memory <INDEX>. <INDEX> is a BYTE index in the FRAM strogae starting with 0.				
SET FRAMDBL	ASCII WRITE COMMAND	#SFRAMDBL:<INDEX>,<DOUBLEVALUE><CR> Result: #SFRAMDBL:OK<CR> or #SFRAMDBL:ERR<CR>	ASCII	YES
	INDEX	400		
	DOUBLEVALUE	3,1415926		
	TX	#255,SFRAMDBL:400,3.1415926<CR>		
	RX	#255,SFRAMDBL:OK<CR>		
Writes a new DOUBLE value (8 byte) into FRAM memory <INDEX>. <INDEX> is a BYTE index in the FRAM strogae starting with 0.				

You can also access this FRAM via MODBUS. For that use the UnitID 2 for accessing the FRAM with READ HOLDING REGISTER and WRITE SINGLE REGISTER or WRITE MULTIPLE REGISTERS commands.
Every register will store two bytes of the FRAM. Again index 0 will be the first two bytes you can use in the FRAM.

To detect the type and used bytes of the FRAM from system in MODBUS use this registers:

FRAM						
GET FRAM TYPE	3x65224 4x65224 I:65223	2,0x0002 B:00 02			UINT16 R/O	
		FRAM size & type:FM25L16 2kB				
Returns the current ytpe of the FRAM and its total size =2:FM25L16B_G, 2kB =64:FM25V05, 64kB =128:FM25V10, 128kB						
GET FRAM USED BYTES	3x65225 4x65225 I:65224	166,0x00A6 B:00 A6			UINT16 R/O	
		FRAM used bytes:166				
Returns the amount of used bytes from system in FRAM						

6.2.6 Execute factory reset

In ASCII you can send the command FRST to restore all values in the FRAM to the system default values.
But you have to disconnect the IoT controller from power, wait a little bit, and then reconnect the device to power, so that all settings can be restored properly!

FACTORY RESET	ASCII WRITE COMMAND	#FRST<CR> Result: #OK<CR>	ASCII	NO
	TX	#255,FRST<CR>		
	RX	N/A		
Executes a factory reset of the module				

You can do the same with MODBUS using a coil or a register. Again you have to re-power the device. A reboot of the LINUX alone has no effect!

FACTORY RESET	1x65535 2x65535 I:65534	0,0x00 B:00		1:PERFORM FACTORY RESET	BIT R/W	NO
Performs a factory reset of all internal saved parameters						
FACTORY RESET	3x65535 4x65535 I:65534	0,0x0000 B:00 00		1:PERFORM FACTORY RESET	UINT16 R/W	NO
Performs a factory reset of all internal saved parameters						

6.2.7 Additional WATCHDOG for LINUX

The co-processor offers an additional watchdog for LINUX. This watchdog must be set by your software cyclically. In ASCII use command WD:<WDTime> to set an interval in Milliseconds. If your software do not send this command within the defined period again, the ARM Co-processor will reset the Raspberry Pi Core, this leads to a reboot of the device!

WATCHDOG TIMER	ASCII WRITE COMMAND	#WD:<WDTIME><CR> Result: #OK<CR>	ASCII	YES
	WDTIME	0		
	TX	#255,WD:0<CR>		
	RX	#255,OK<CR>		
Enables or disables the WATCHDOG Timer for the Raspberry Pi module. WDTIME: 1..3600000: Time for Watchdog in Milliseconds (Maximum 60 Minutes) =0: No Watchdog is generated HINT: The Watchdog is internally handled every 10ms, so every value below 10 will reset immediately the Raspberry Pi computer.				

The same watchdog is available for MODBUS. A value of 0 in both cases will deactivate the watchdog functionality.

RASPBERRY PI WATCHDOG TIMER	3x65223 4x65223 I:65222	0,0x0000 B:00 00		50	UINT16 R/W	NO
		remaining watchdog time in 10ms:0 -> 0,000s				
Enables or disables the WATCHDOG Timer for the Raspberry Pi module. 1..65535: Time for Watchdog in x10 Milliseconds (Maximum 655,35 seconds) =0: No Watchdog is generated						

6.2.8 INIT VALUES & COMMUNICATION WATCHDOG for IOs

The co-processor has a build in functionality to set the digital and analog outputs to a defined state after a power on and if defined an IO watchdog event.

Therefore you can define for every digital output configuration values (1 or 0), but this affects also the diagnostic features of the digital outputs. For the universal analog IOs you can define for every configured analog output a value which is outputted at system startup and in case of an IO watchdog event. See the specific chapters, how to define this init & watchdog values for the digital outputs and universal analog IOs.

If the watchdog timer is set to 0, only after system startup the configured values are set on the outputs.

But if you set with the ASCII command SIOWATCHDOG a new IO watchdog time, the outputs will be set to the configuration state if there is no ASCII communication and no MODBUS communication for this time span!
With the command GIOWATCHDOG you can read the current settings. A 0 value deactivates the IO watchdog feature.

SET IO WATCHDOG TIMER	ASCII WRITE COMMAND	#SIOWATCHDOG:<IOWDTIME><CR> Result: #OK<CR>	ASCII	YES
	IOWDTIME	0		
	TX	#255,SIOWATCHDOG:0<CR>		
	RX	#255,OK<CR>		
Sets a new time for the internal IO WATCHDOG Timer. <IOWDTIME> is a time in 100ms. =0: No IO Watchdog is used HINT: The Watchdog is internally handled every 100ms, if the Timer reaches 0, all internal IOS will be set to a preconfigured state. Every ASCII command or MODBUS request will reset this timer.				
GET IO WATCHDOG TIMER	ASCII READ COMMAND	#GIOWATCHDOG<CR> Result: #GIOWATCHDOG::<IOWDTIME><CR>	ASCII	
	TX	#255,GIOWATCHDOG<CR>		
	RX	#255,GIOWATCHDOG:0,0x0<CR>		
Returns the actual time for the internal IO WATCHDOG Timer. <IOWDTIME> is a time in 100ms. =0: No IO Watchdog is used HINT: The Watchdog is internally handled every 100ms, if the Timer reaches 0, all internal IOS will be set to a preconfigured state. Every ASCII command or MODBUS request will reset this timer.				

The same watchdog is available for MODBUS.

MODBUS WATCHDOG						
MODBUS WATCHDOG TIME	3x65222 4x65222 I:65221	0,0x0000 B:00 00		50	UINT16 R/W	NO
		Actual watchdog time in 1/100s:0 -> 0,0s				
Writing a value onto this register defines a new time for the internal communication watchdog timer. The value is a timespan in 1/100s. =0: The communication watchdog is disabled =1..65535: Communication watchdog will be trigegred after x 1/100s pause on communication line In case of an communication watchdog, the module sets all outputs to the states defined in the configuration output registers Reading this register will return the current stored time from the internal FRAM						

7 RESI-T4-xxx IoT Controller

7.1 Basic functionality of T4 IoT family

Our RESI-T4 IoT controller are based on the Raspberry® Pi 4 module.
In general the Raspberry Pi 4 Model B offers the following features:

- Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit
- SoC @ 1.5GHz
- Memory: 1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
- Connectivity: .4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE
- 1xGigabit Ethernet
- 2 × USB 3.0 ports
- 2 × USB 2.0 ports.
- Video & sound: 2×micro HDMI ports (up to 4Kp60 supported)
- 4-pole stereo audio and composite video port
- Multimedia: H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
- SD card slot: Micro SD card slot for loading operating system
- LINUX® pre-installed

We added the following features to this board:

- Industrial grade housing: 4MU 73x110x62mm
- Mounting on DIN rail or on-wall
- Industrial grade wide range power supply: 12-48Vdc
- 4 status LEDs and 8pin DIP switch for software usage
- Maximum of three serial interfaces: RS232 or RS485
- Optional one KNX interface
- All serial interfaces appear as native serial lines in LINUX (dev/ttyACM0- dev/ttyACMn)
- No need for specific LINUX kernel drivers or real-time OS
- RS485 direction switching is done in hardware by Co-processor
- ARM® Co-processor for additional features:
  - real time clock with backup capacitor
  - Unique 96-Bit serial number
  - 2kB ferromagnetic RAM
  - handles the DIP switch and the LEDs
- ARM Co-processor is connected to LINUX via serial interface dev/ttyACM0 and simple ASCII commands

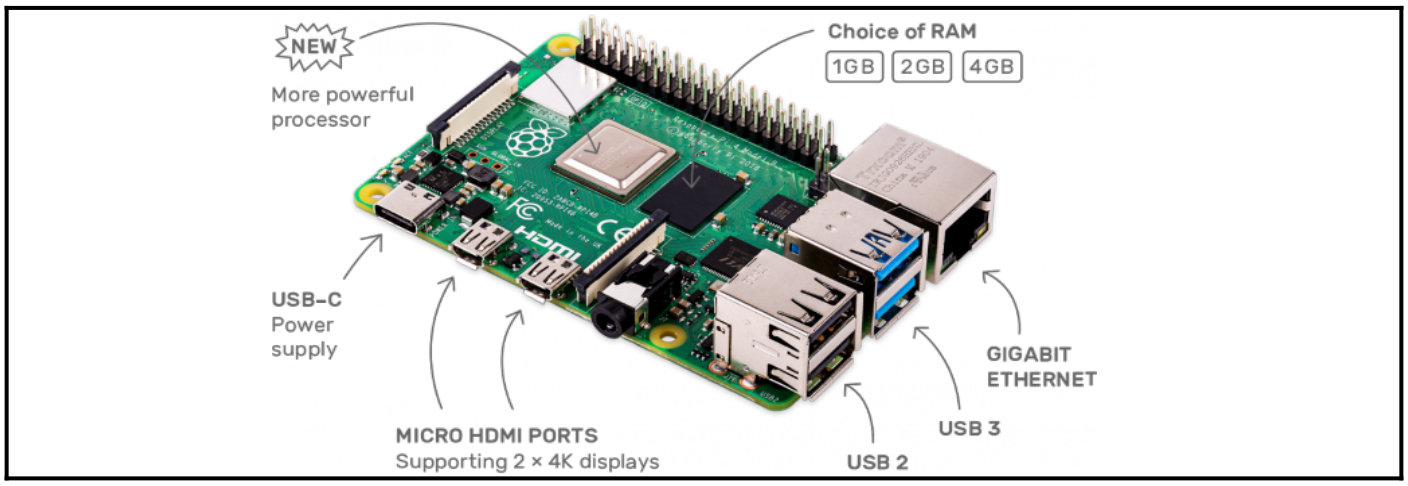


Figure: Raspberry® Pi 4

7.2 RESI-T4-Z basic module

This version offers an industrial grade controller version, just with the Raspberry Pi4 Model B board and a robust 12-48V= power supply.

Figure: Our RESI-T4-Z IoT Controller

7.2.1 Technical specification

Beside the basic technical data, which fulfill all of our T4 IoT Controller, this specific controller meets the following technical specifications:

Power consumption	<25W	
Product housing	T4-XT4	
Product weight	RESI-T4-Z	175g
	RESI-T4-N-CAN	193g
	RESI-T4-N-CFD	193g

No LEDs and no DIP switch
No ARM Co-processor

7.2.2 Additional terminals or functionalities

This IoT controller has no additional terminals or functionalities.

7.2.3 Connection diagram

7.2.3.1 Cabling of the power supply and the Ethernet

This controller offers only terminals for power supply and a RJ45 connector for Ethernet. But the standard Raspberry Pi 4 connectors are also available: 2xUSB 2.0, 2xUSB 3.0, 2xMicro HDMI, 1xAudio, 1xSD-CARD Slot.

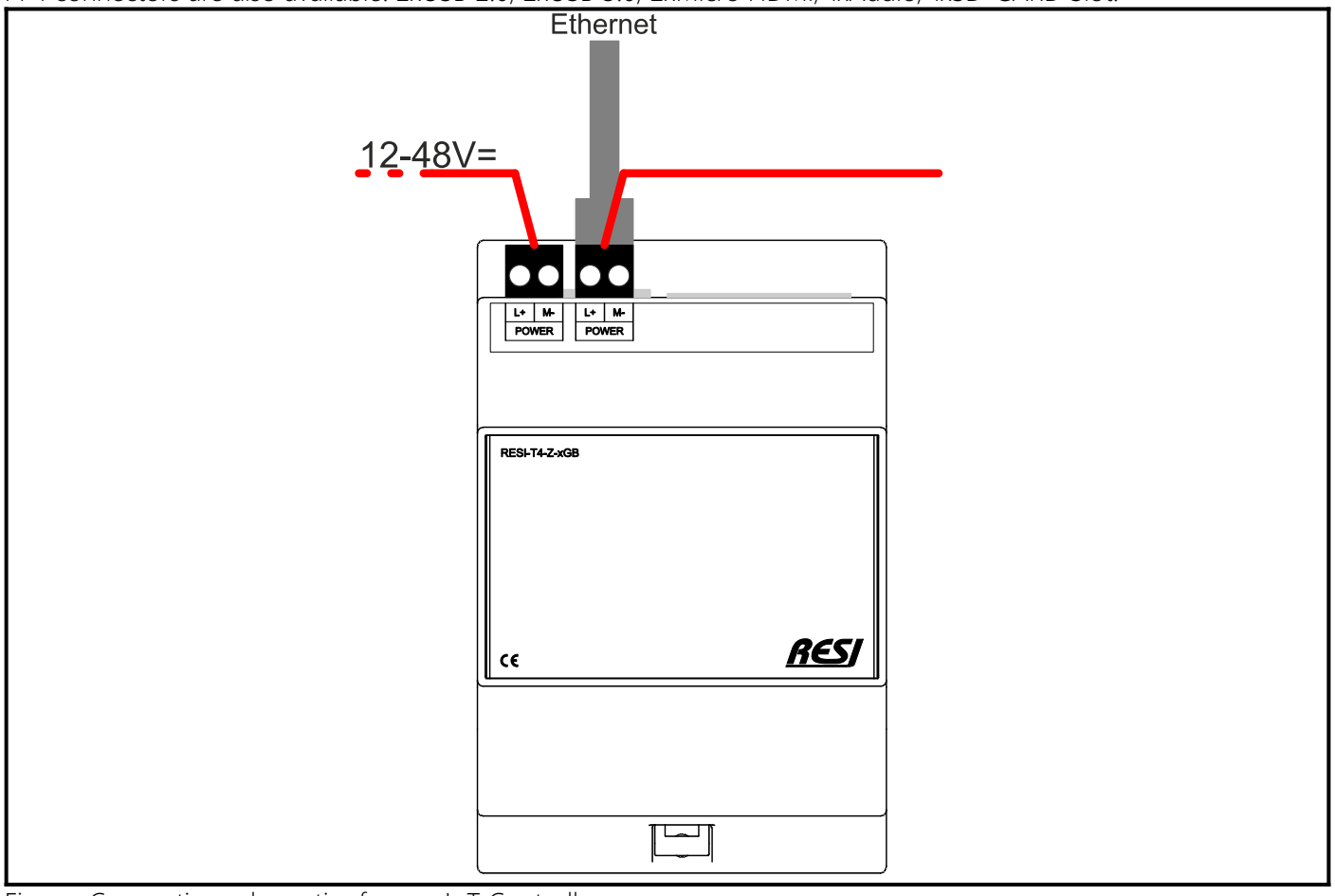


Figure: Connection schematics for our IoT Controller

7.3 RESI-T4-xxx-CAN/CAN FD IoT Controller

All of our RESI-T4 controller can also have a build in CAN 2.0 or CAN FD interface. This CAN interface can be used by open source Raspberry Pi CAN software using SPI0.

7.3.1 Technical specification

Beside the basic technical data, which fulfill all of our T4 IoT controller, this specific controller meets the following technical specifications:

CAN interface	
RESI-T4-xxx-CAN	CAN 2.0 interface
RESI-T4-xxx-CFD	CAN FD interface

CAN interface is connected to the SPI0 interface of the Raspberry PI

7.3.2 Additional terminals or functionalities

This module has an additional connector for CAN interface on the side of the Ethernet connector of the Raspberry Pi 4.

CAN/CAN FD	CAN 2.0 or CAN FD interface	
	Pin 1:	H: CAN HIGH signal
	Pin 2:	L: CAN LOW signal
	Pin 3:	G: CAN Ground signal
	Terminal type:	RM3.5

7.3.3 Connection diagram

7.3.3.1 Additional cabling of the CAN/CAN FD interface

In addition to the standard connectors described above, this IoT controller offers a CAN or CAN FD connector.

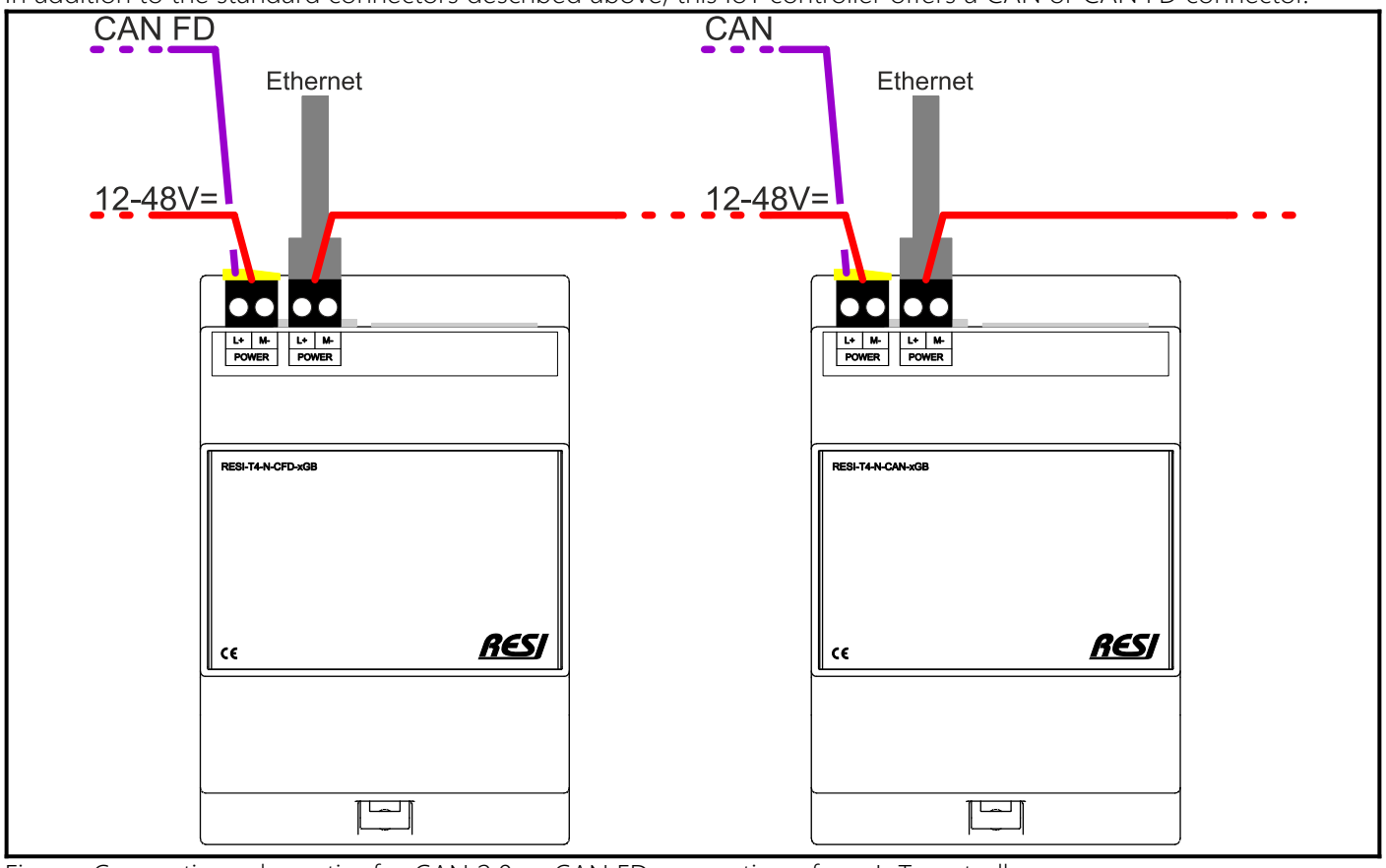


Figure: Connection schematics for CAN 2.0 or CAN FD connection of our IoT controller

7.4 RESI-T4-A,B,C,D with serial interfaces RS232 or RS485

We support various versions of our T4 IoT controller with build in serial interfaces.

- RESI-T4-A-xGB: 3xRS485
- RESI-T4-B-xGB: 2xRS485 and 1xRS232
- RESI-T4-C-xGB: 1xRS485 and 2xRS232
- RESI-T4-D-xGB: 3xRS232

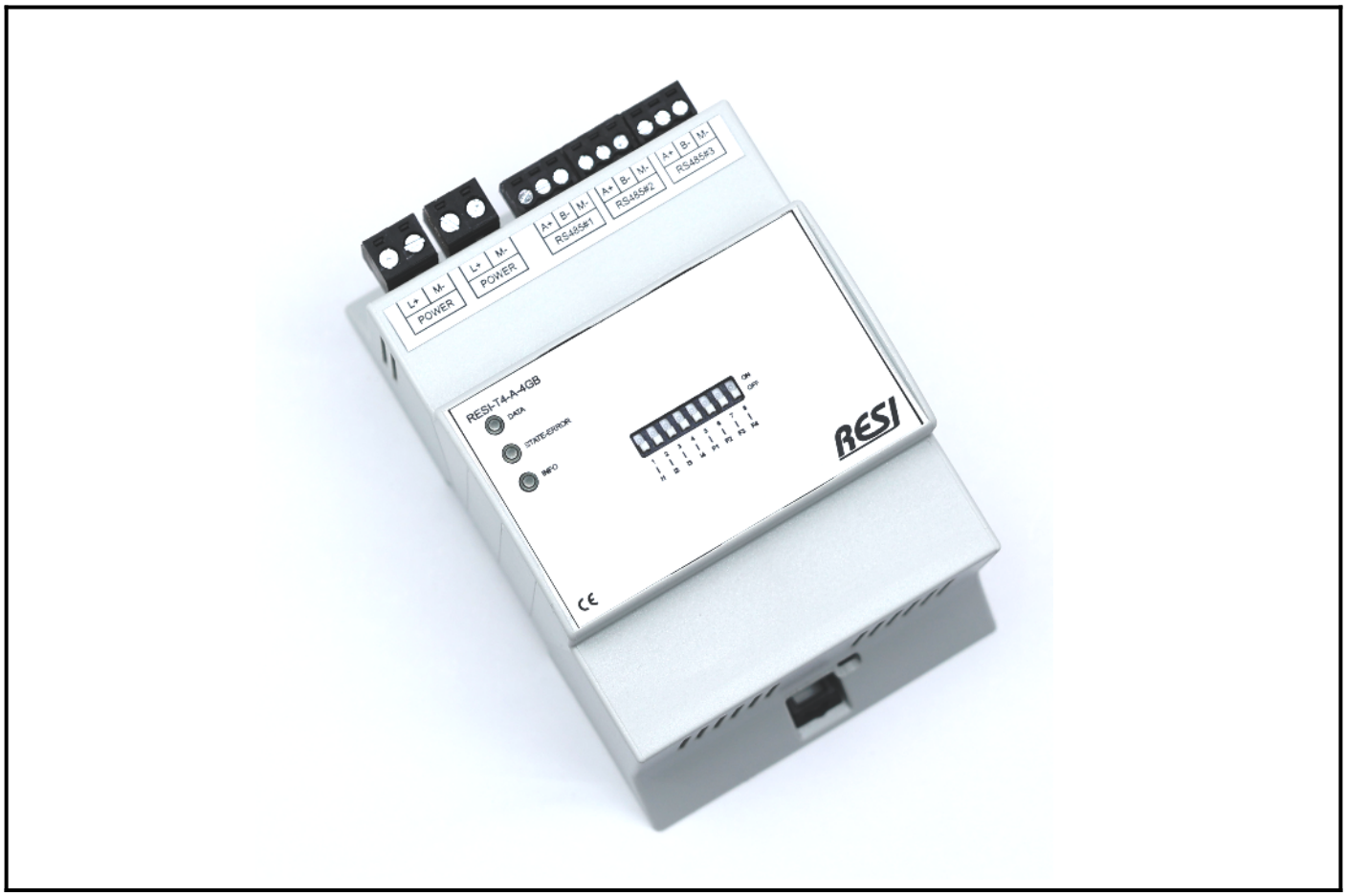


Figure: Our RESI-T4-A IoT controller with three serial RS485 interfaces

7.4.1 Technical specification

Beside the basic technical data, which fulfill all of our T4 IoT controller, this specific controller meets the following technical specifications:

Power consumption	<25W	
Product housing	T4-XT4	
Product weight	RESI-T4-A,B,C,D	192g
	RESI-T4-A,B,C,D-CAN	210g
	RESI-T4-A,B,C,D-CFD	210g

4 LEDs and 8 pin DIP switch
ARM co-processor with real time clock+backup capacitor, 2kB ferromagnetic RAM

Serial interfaces	
RESI-T4-A	3xRS485, automatic direction control
RESI-T4-B	2xRS485, automatic direction control
	1xRS232
RESI-T4-C	1xRS485, automatic direction control
	2xRS232
RESI-T4-D	3xRS232

Serial interfaces are connected as USB serial lines to LINUX with dev/ttyACMx in the Raspberry PI

7.4.2 Additional terminals or functionalities

This module has three additional connectors for serial interfaces.

Depending on IoT controller:	Up to 3xRS485 or up to 3xRS232

RS485#?	RS485 serial interface	
	Pin 1:	A+: RS485 DATA+ signal
	Pin 2:	B-: RS485 DATA- signal
	Pin 3:	M-: RS485 ground signal
	Terminal type:	RM3.5

RS232#?	RS232 serial interface	
	Pin 1:	TX: RS232 DATA+ signal
	Pin 2:	RX: RS232 DATA- signal
	Pin 3:	M-: RS232 ground signal
	Terminal type:	RM3.5

7.4.3 Connection diagram

7.4.3.1 RESI-T4-A additional cabling

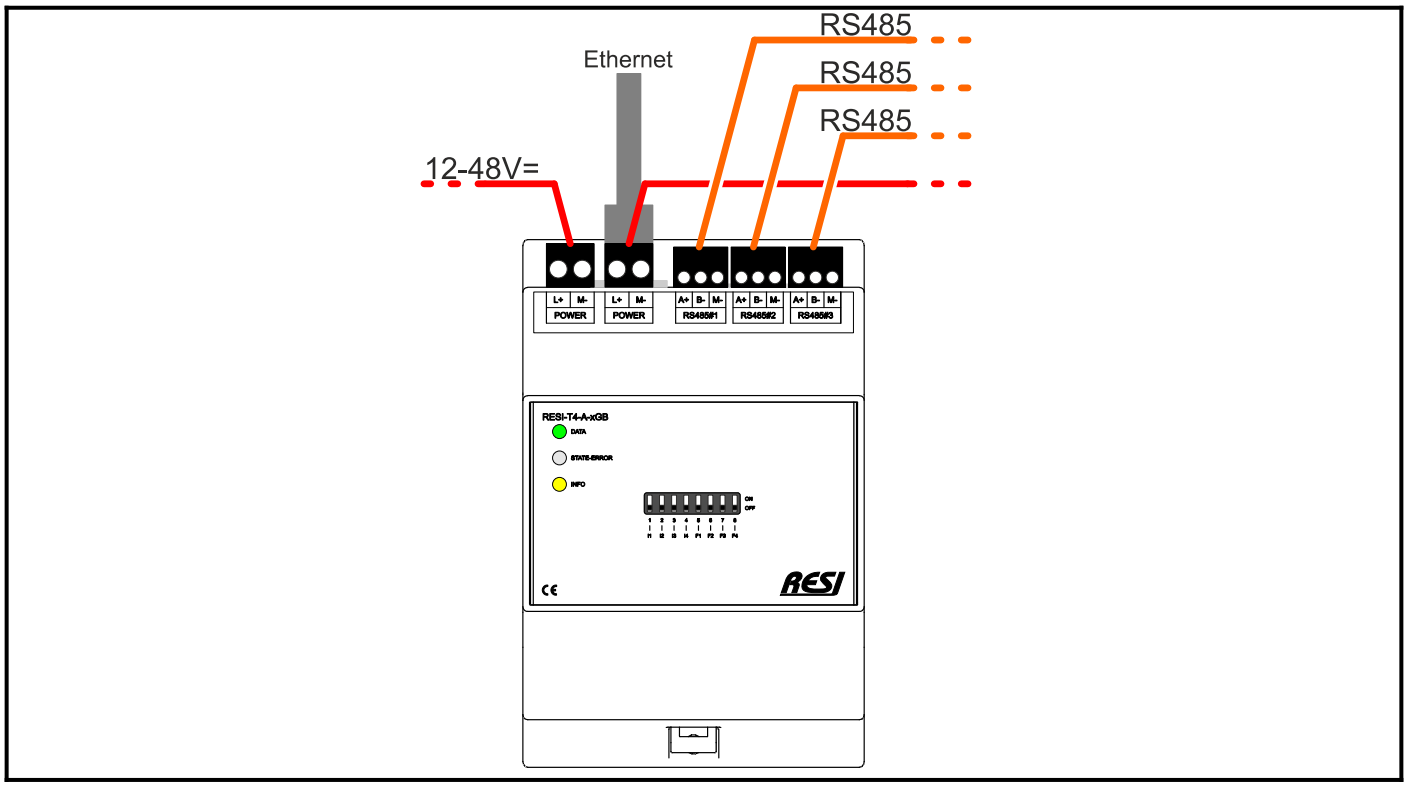


Figure: Connection schematics for our IoT controller

7.4.3.2 RESI-T4-B additional cabling

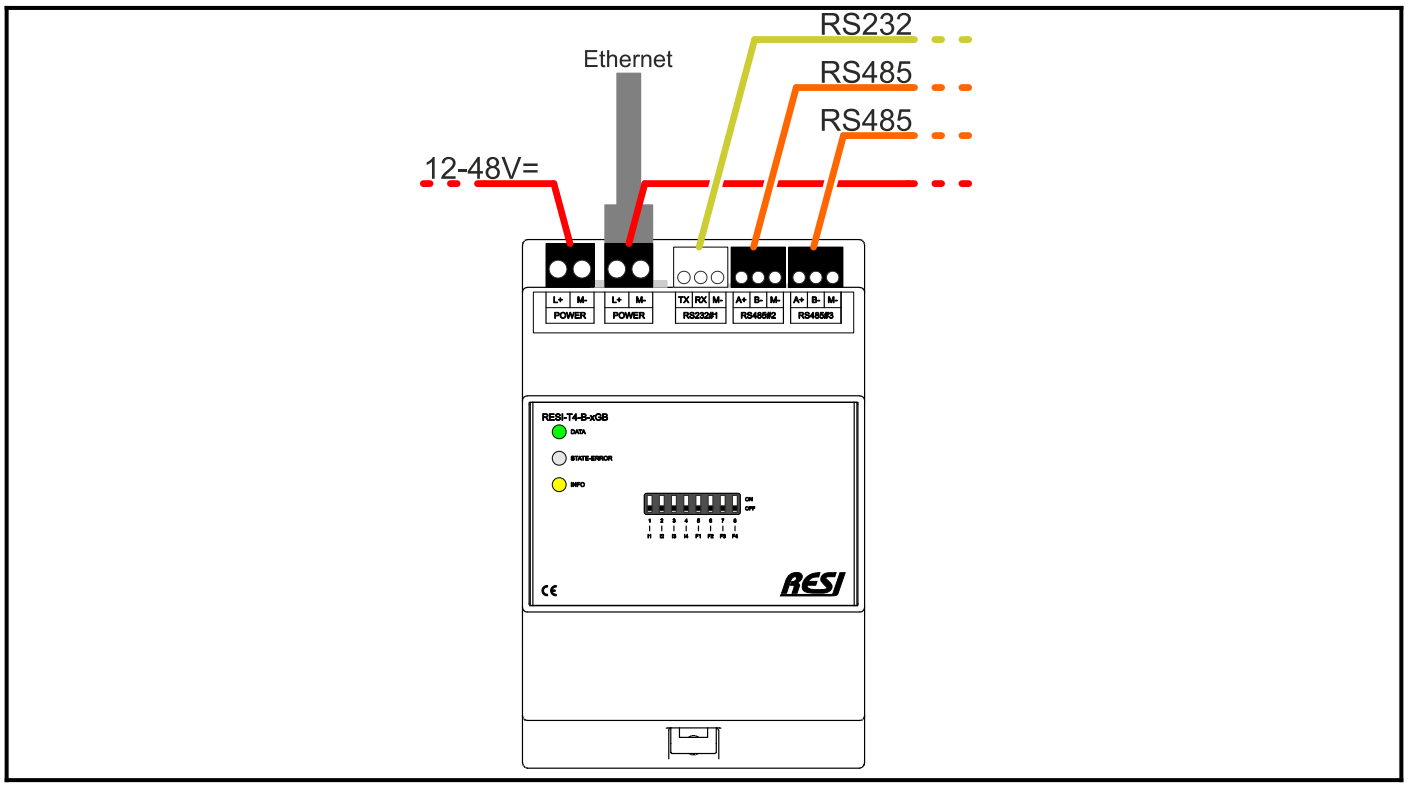


Figure: Connection schematics for our IoT controller

7.4.3.3 RESI-T4-C additional cabling

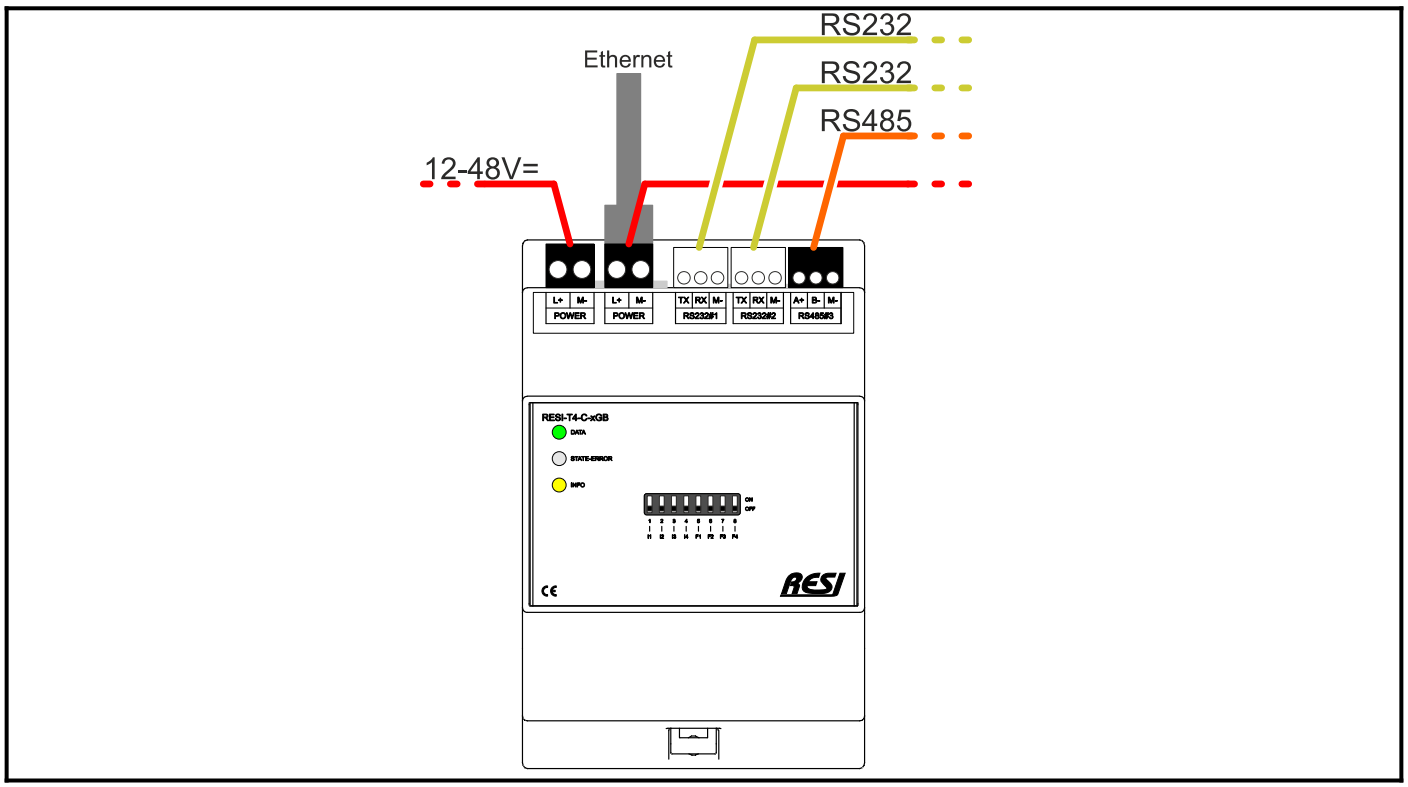


Figure: Connection schematics for our IoT controller

7.4.3.4 RESI-T4-D additional cabling

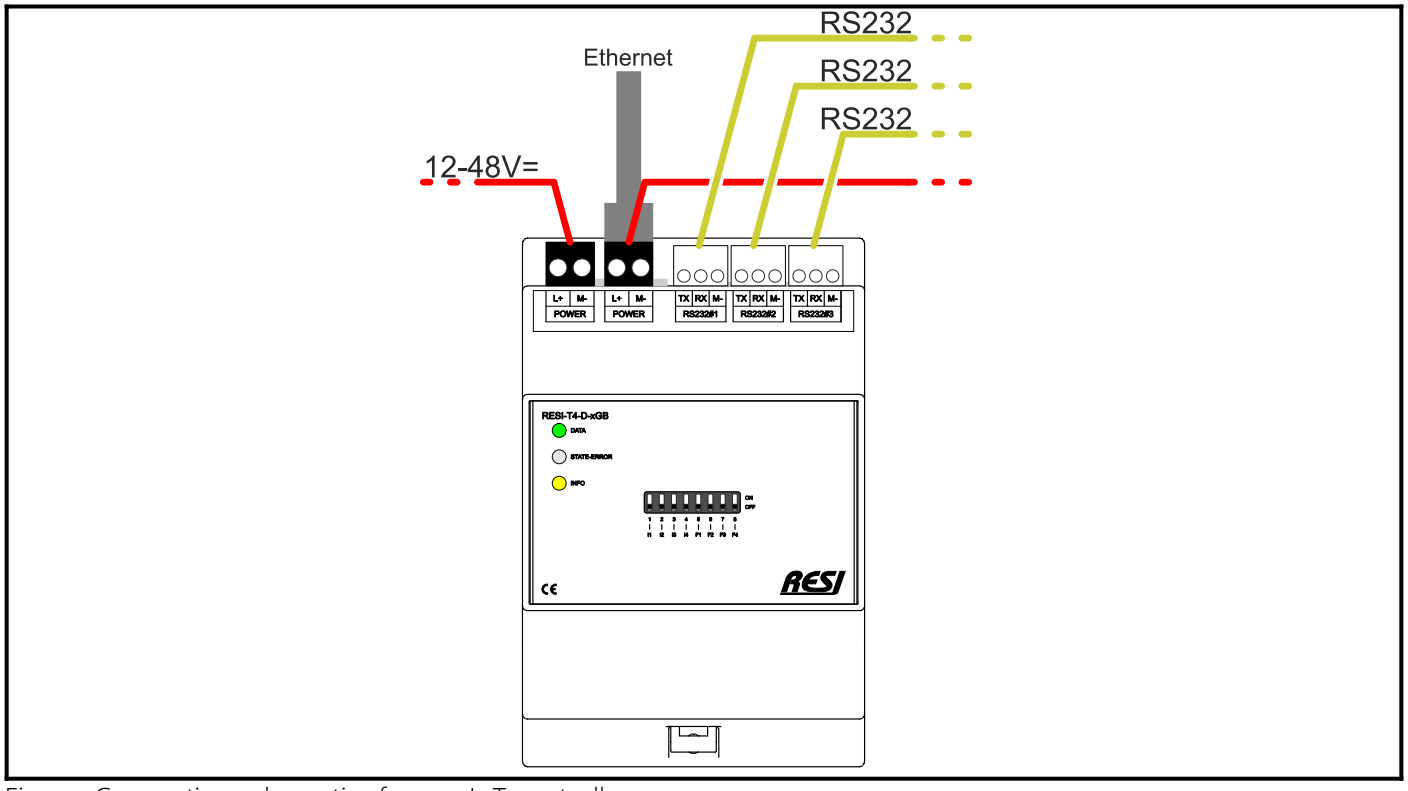


Figure: Connection schematics for our IoT controller

7.5 RESI-T4-KA,KB,KC with KNX interface+RS232 or RS485

We support various versions of our T4 IoT controller with build in serial interfaces.

- RESI-T4-KA-xGB: 1xKNX, 2xRS485
- RESI-T4-KB-xGB: 1xKNX, 1xRS485 and 1xRS232
- RESI-T4-KC-xGB: 1xKNX, 2xRS232

Figure: Our RESI-T4-KA IoT controller with one KNX and two serial RS485 interfaces

7.5.1 Technical specification

Beside the basic technical data, which fulfill all of our T4 IoT controller, this specific controller meets the following technical specifications:

Power consumption	<25W	
Product housing	T4-XT4	
Product weight	RESI-T4-KA,KB,KC	194g
	RESI-T4-Kx-CAN	210g
	RESI-T4-Kx-CFD	210g

4 LEDs and 8 pin DIP switch
ARM co-processor with real time clock+backup capacitor, 2kB ferromagnetic RAM

Serial interfaces	
RESI-T4-KA	1xKNX
	2xRS485, automatic direction control
RESI-T4-KB	1xKNX
	1xRS485, automatic direction control
	1xRS232
RESI-T4-KC	1xKNX
	2xRS232

Serial interfaces are connected as USB serial lines to LINUX with dev/ttyACMx in the Raspberry PI

7.5.2 Additional terminals or functionalities

This module has three additional connectors for serial interfaces.
Depending on IoT controller: 1xKNX and up to 2xRS485 or up to 2xRS232

KNX	KNX interface	
	Pin 1:	K+: KNX+ signal (RED)
	Pin 2:	K-: KNX- signal (BLACK)
	Terminal type:	RM3.5
	To use the KNX interface you have to use an external KNX power supply!	

RS485#?	RS485 serial interface	
	Pin 1:	A+: RS485 DATA+ signal
	Pin 2:	B-: RS485 DATA- signal
	Pin 3:	M-: RS485 ground signal
	Terminal type:	RM3.5

RS232#?	RS232 serial interface	
	Pin 1:	TX: RS232 DATA+ signal
	Pin 2:	RX: RS232 DATA- signal
	Pin 3:	M-: RS232 ground signal
	Terminal type:	RM3.5

7.5.3 Connection diagram

7.5.3.1 RESI-T4-KA additional cabling

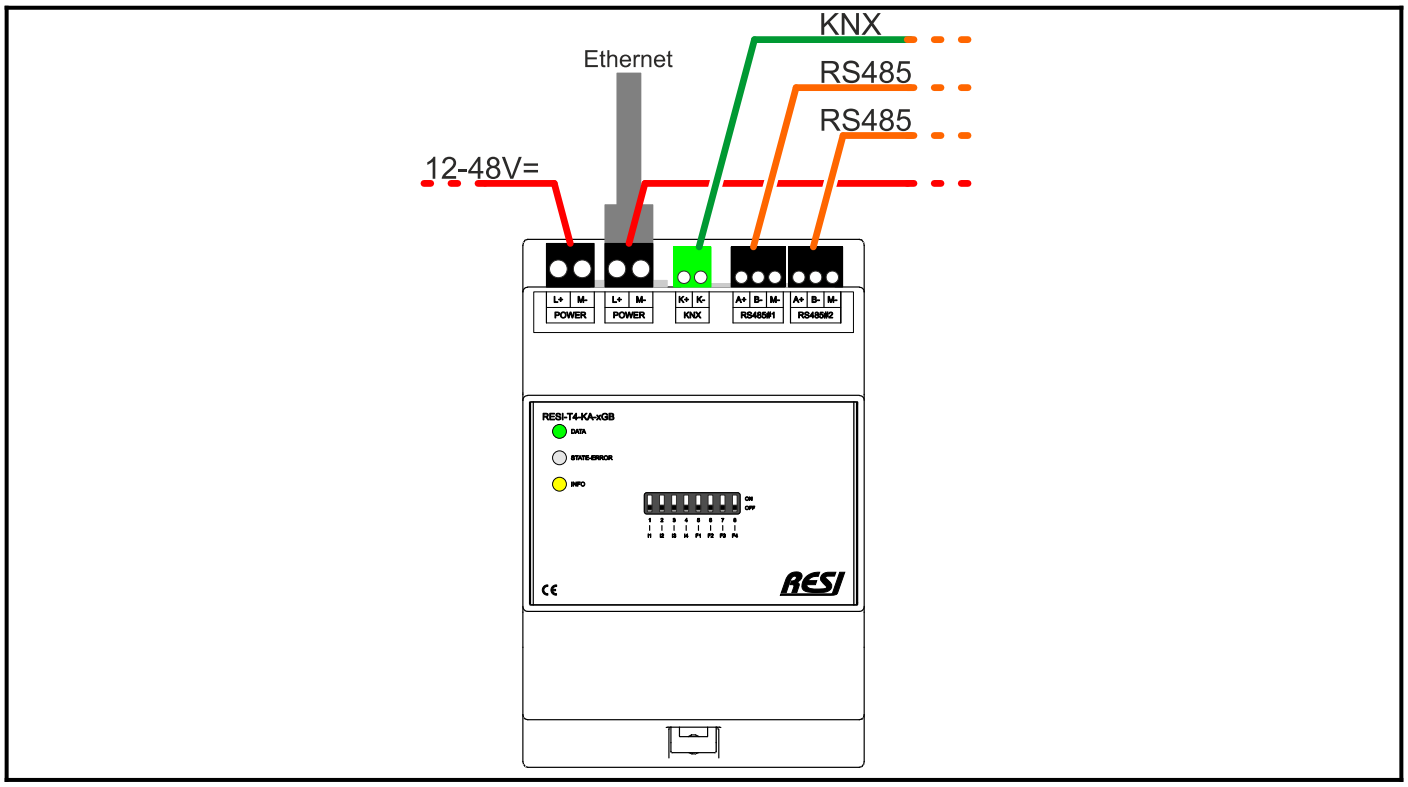


Figure: Connection schematics for our IoT controller

7.5.3.2 RESI-T4-KB additional cabling

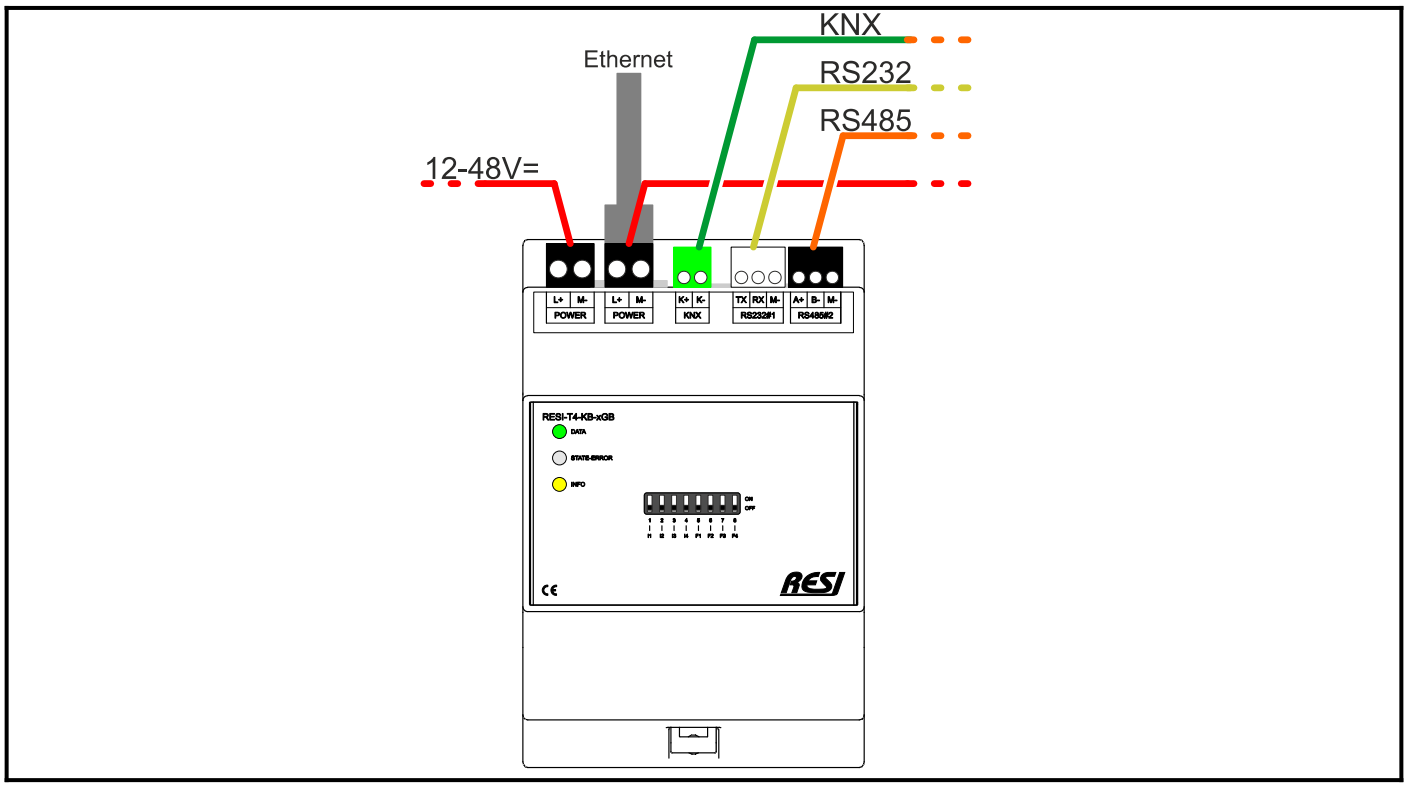


Figure: Connection schematics for our IoT controller

7.5.3.3 RESI-T4-KC additional cabling

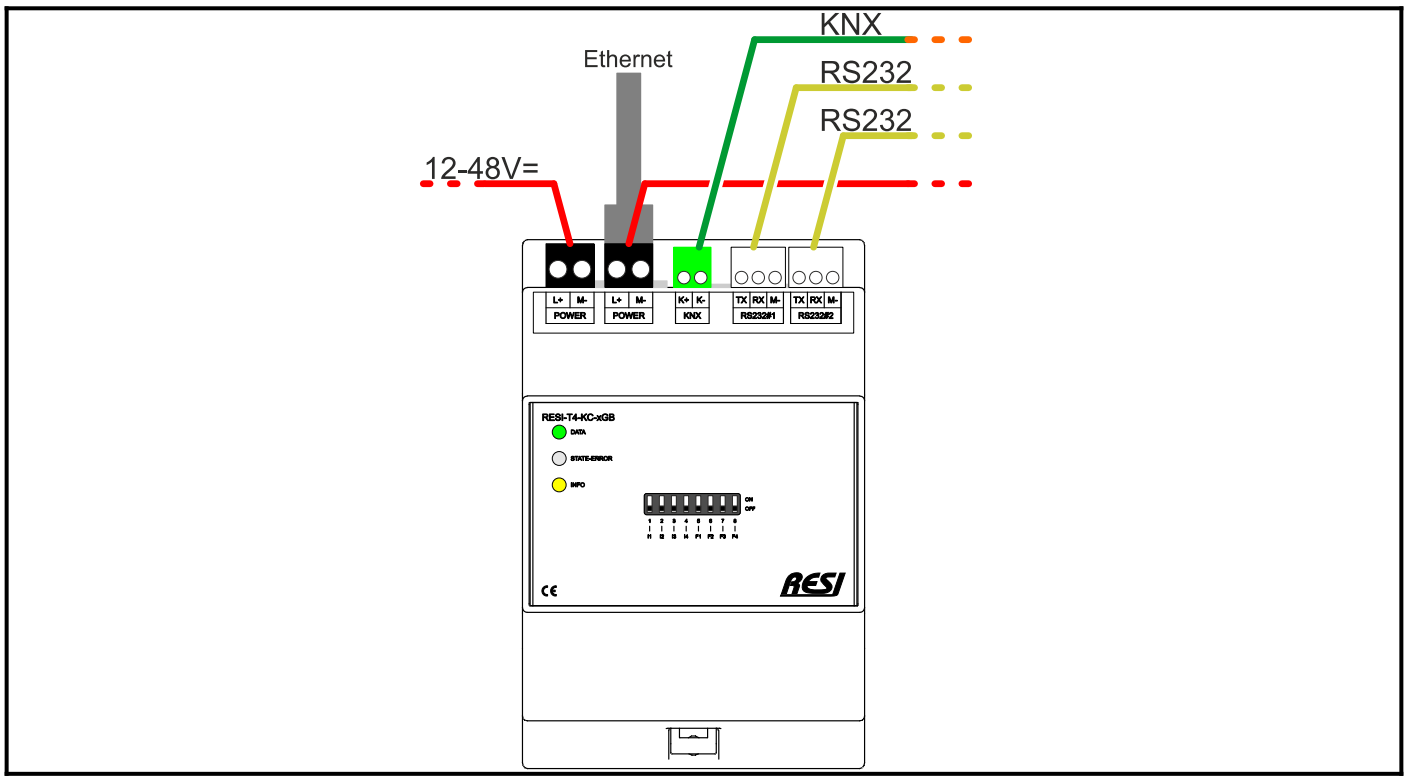


Figure: Connection schematics for our IoT controller

8 RESI-C4-xxx IoT controller

8.1 Basic functionality of C4 family

Our RESI-C4 IoT controller are based on the Raspberry® Pi Compute Module 4.
In general the Raspberry Pi Compute Module 4 offers the following features:

- Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit
- SoC @ 1.5GHz
- Memory: 1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
- 1xGigabit Ethernet
- 2 × USB 2.0 ports.
- Video & sound: 1×micro HDMI ports (up to 4Kp60 supported)
- Multimedia: H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
- SD card slot: Micro SD card slot for loading operating system
- LINUX® pre-installed

We added the following features to this board:

- Additional integrated digital and analog inputs and output
- Industrial grade housing: 4MU (73x110x62mm) or 8MU (143x110x62mm) or 12MU (213x110x62mm)
- Mounting on DIN rail or on-wall
- Industrial grade wide range power supply: 12-48Vdc
- 4 status LEDs and 8pin DIP switch for software usage
- One serial interface: RS485
- RS485 direction switching is done in hardware by Co-processor
- Serial interface appears as native serial line in LINUX (dev/ttyACM0- dev/ttyACMn)
- No need for specific LINUX kernel drivers or real-time OS
- ARM® Co-processor for additional features:
  - handles all integrated digital and analog IOs
  - real time clock with backup capacitor
  - Unique 96-Bit serial number
  - 2kB ferromagnetic RAM
  - handles the DIP switch and the LEDs
- ARM Co-processor is connected to LINUX via two independent serial interfaces
  - dev/ttyACM0 for communication with simple ASCII commands
  - dev/ttyACM1 for communication with MODBUS/RTU Master protocol

Figure: Raspberry® Pi Compute Module 4

8.2 RESI-C4-A,-2E,-LTE with serial interface RS485

We support various versions of our C4 IoT controller with build-in additional IOs and other features like 2nd Ethernet interface or LTE modem.

- RESI-C4-A-xGB: 1xRS485, 1xMicro HDMI
- RESI-C4-A-2E-xGB: 1xRS485, 1xMicro HDMI, 2nd Ethernet Interface
- RESI-C4-A-LTE-xGB: 1xRS485, 1xMicro HDMI, LTE Modem (Quectel EC25)

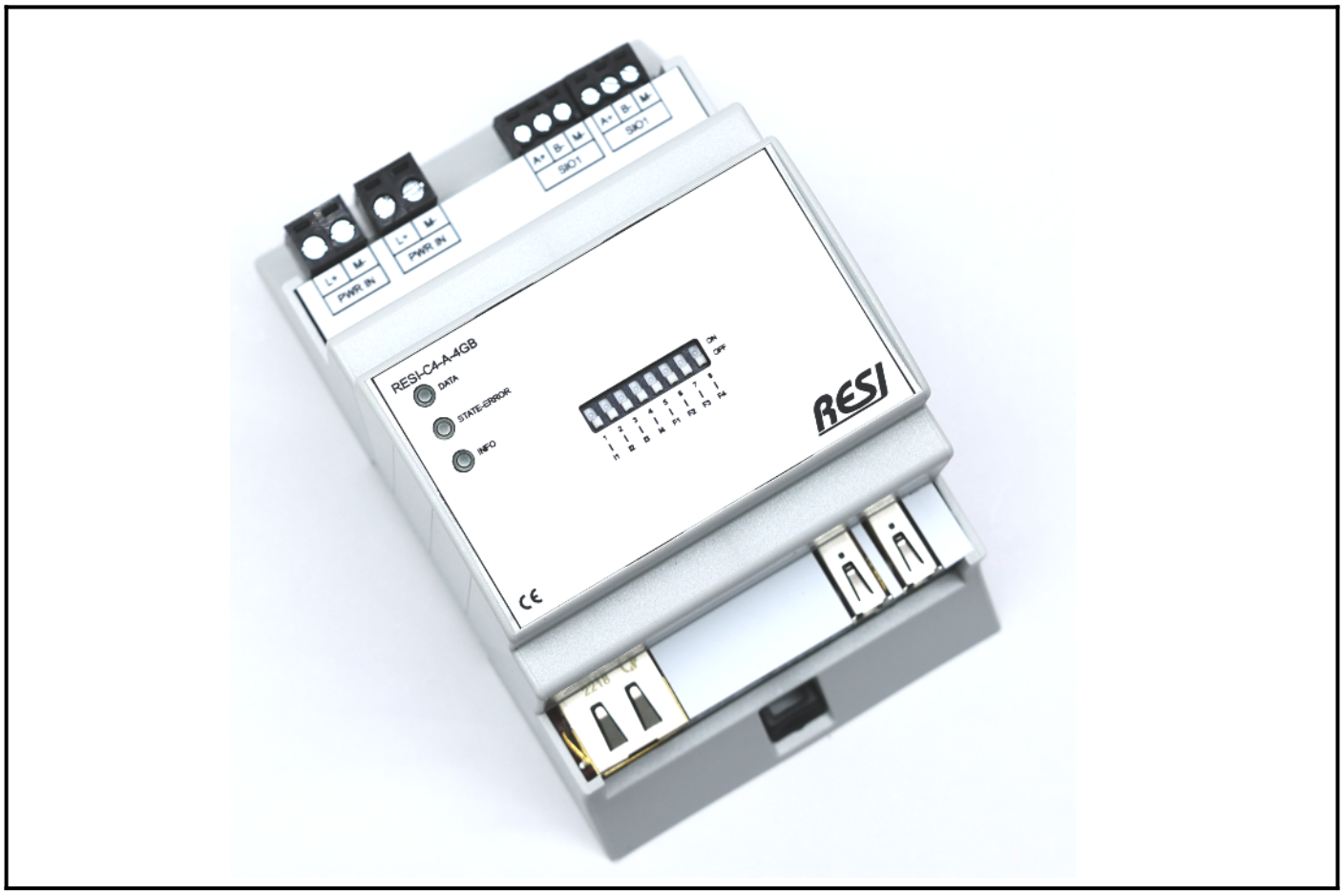


Figure: Our RESI-C4-A IoT controller with one serial RS485 interface and HDMI output

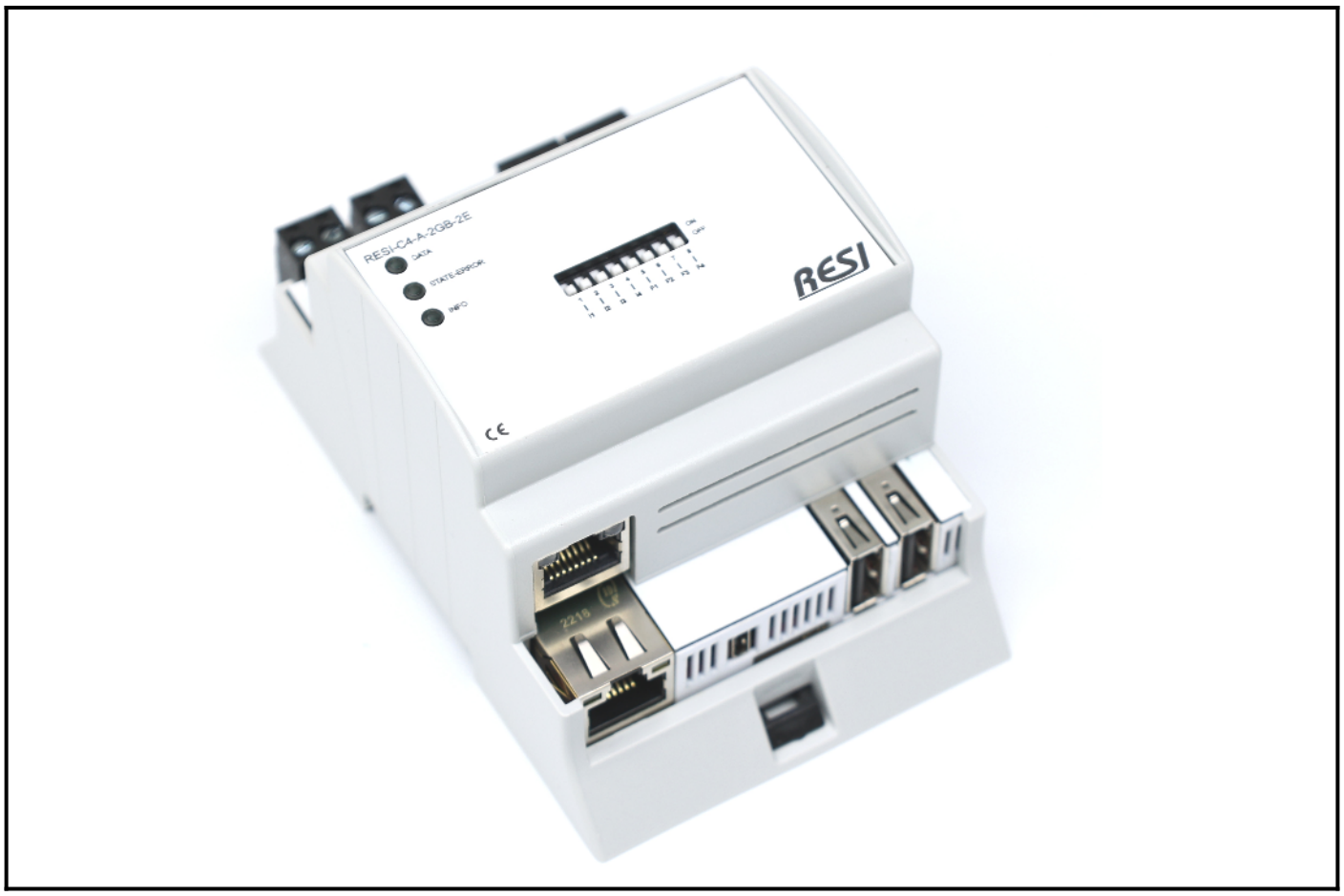


Figure: Our RESI-C4-A-2E IoT controller with 1xRS485 interface, HDMI output and 2nd Ethernet Interface

Figure: Our RESI-C4-A-LTE IoT controller with 1xRS485 interface, HDMI output and LTE modem

8.2.1 Technical specification

Beside the basic technical data, which fulfill all of our C4 IoT controller, this specific controller meets the following technical specifications:

Power consumption	<25W
Product housing	C4-XT4
Product weight	
RESI-C4-A	179g
RESI-C4-A-2E	195g
RESI-C4-A-LTE	201g

4 LEDs and 8 pin DIP switch
ARM co-processor with real time clock+backup capacitor, 2kB ferromagnetic RAM

Serial interfaces	
	1xRS485, automatic direction control
Serial interface is connected as USB serial line to LINUX with dev/ttyACMx in the Raspberry PI	

2nd Ethernet	connected via USB 2.0 to CM4
	native as eth1 in LINUX available

LTE Modem	Quectel EC25
	connected via USB 2.0 to CM4
	native as wwan0 in LINUX available
	after correct configuration and SIM card inserted

8.2.2 Additional terminals or functionalities

This module has one additional connector for serial interface.

	1xRS485

SIO1	RS485 serial interface	
	Pin 1:	A+: RS485 DATA+ signal
	Pin 2:	B-: RS485 DATA- signal
	Pin 3:	M-: RS485 ground signal
	Terminal type:	RM3.5

8.2.3 Connection diagram

8.2.3.1 RESI-C4-A additional cabling

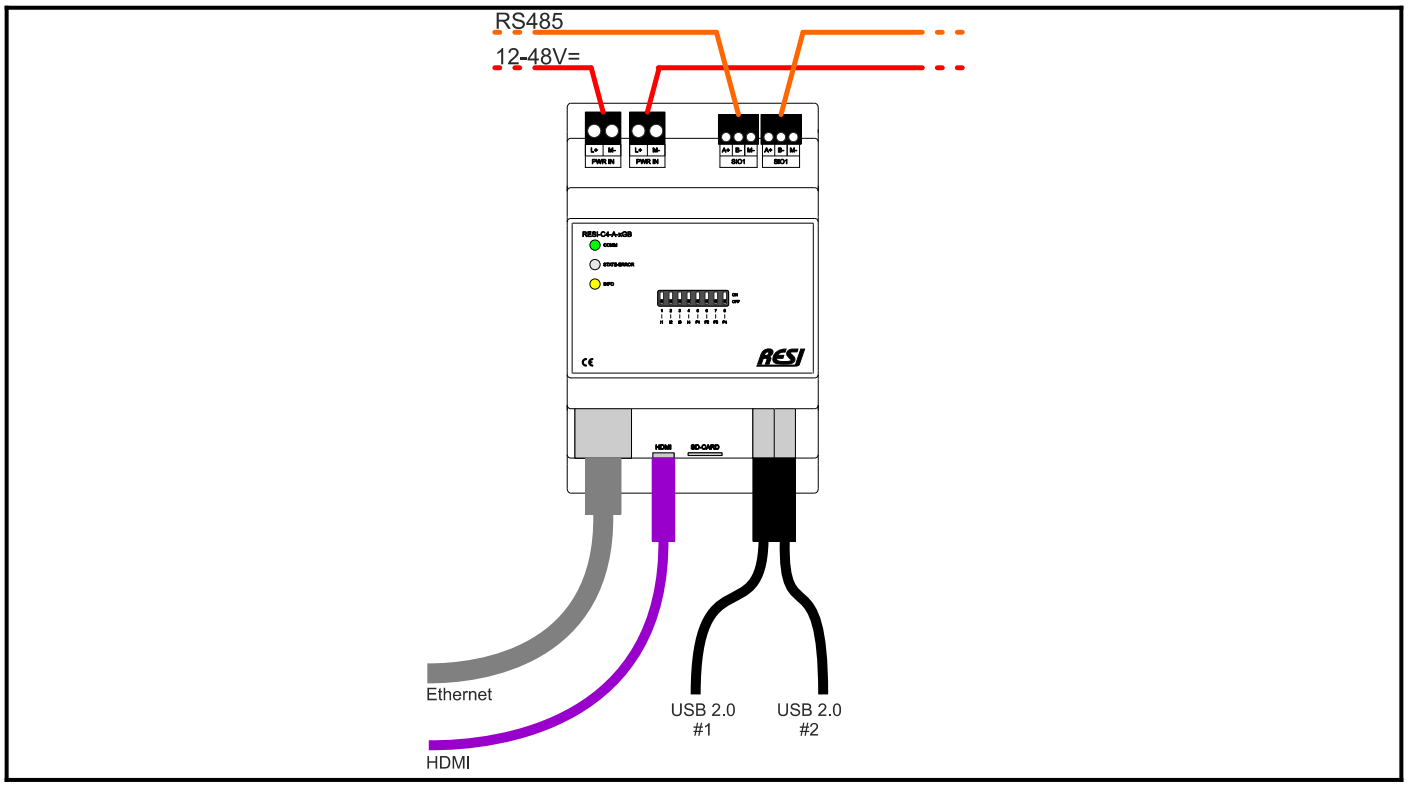


Figure: Connection schematics for our IoT controller

8.2.3.2 RESI-C4-A-2E additional cabling

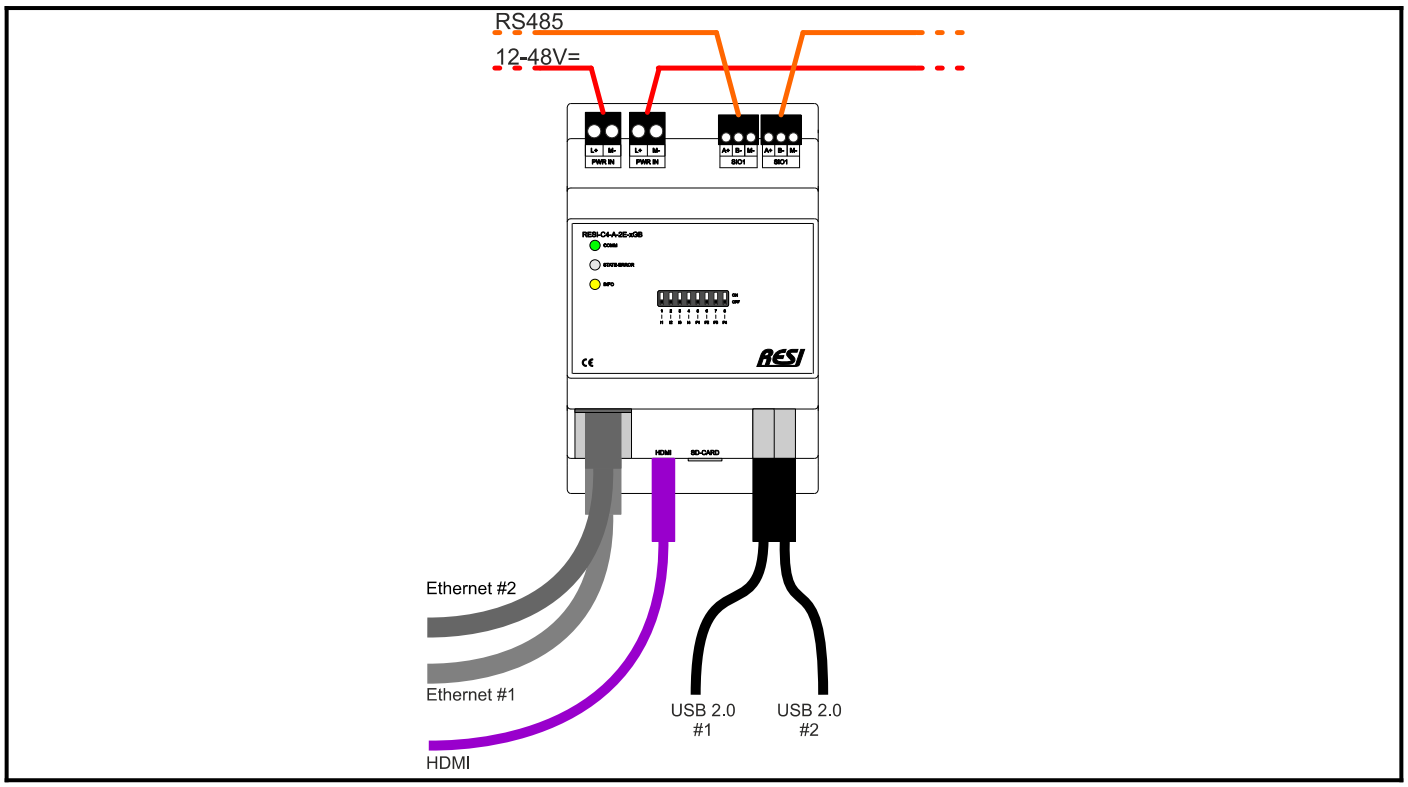


Figure: Connection schematics for our IoT controller

8.2.3.3 RESI-C4-A-LTE additional cabling

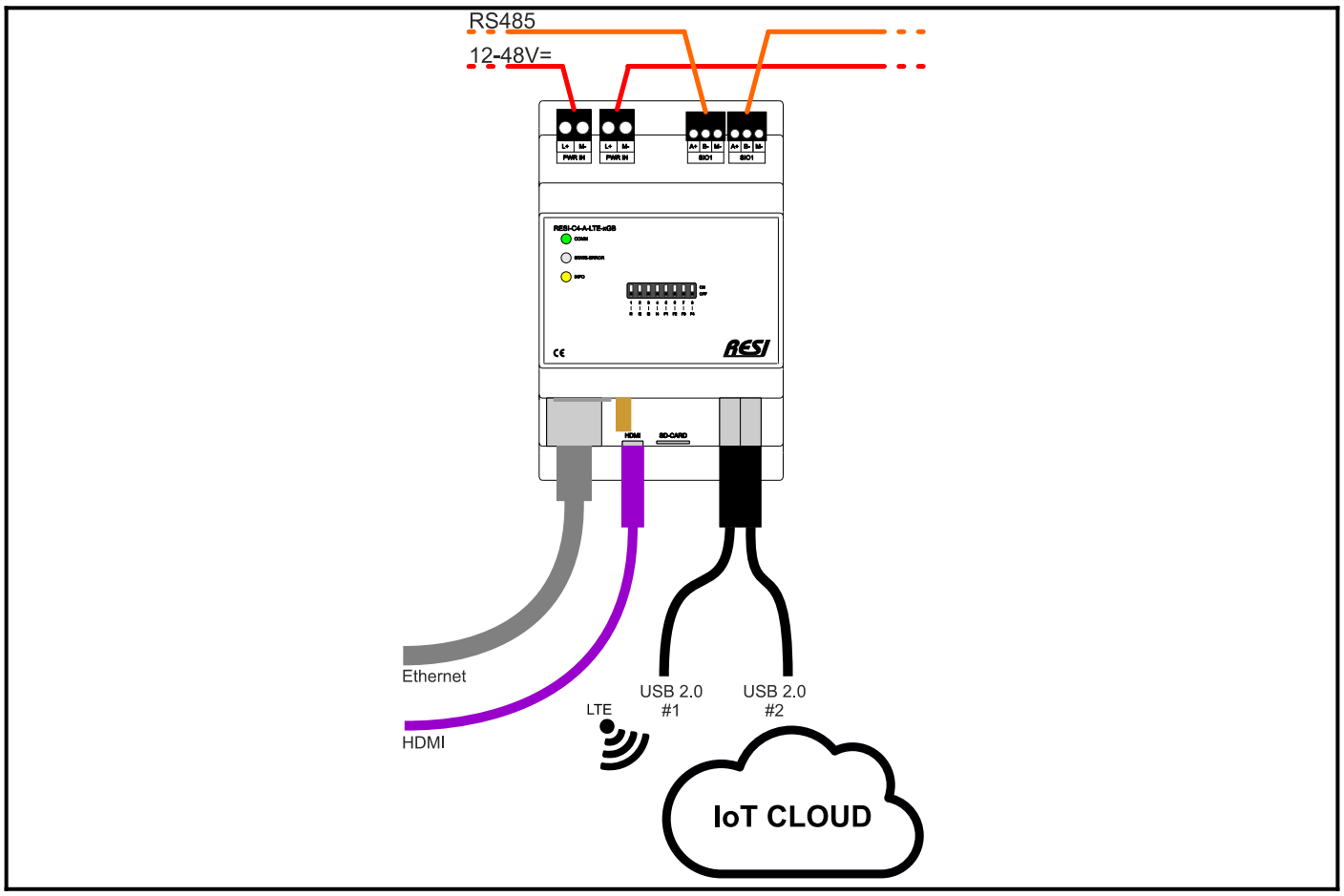


Figure: Connection schematics for our IoT controller

8.3 Which IO types do our RESI-C4 series offer

Our IoT controllers offer various combinations of digital inputs and outputs and universal analog inputs and outputs. Also we offer various types of relay outputs and special modules.

Therefore in this section we explain the basic principal of each IO class.

8.3.1 Digital inputs DC 12-48V=

This input type supports DC signals with 12-48Vdc. It drives the input with max. 1.8mA current. Also the digital inputs are not galvanically insulated from the rest of the controller. So connect the ground of your power supply for the input signal with the ground of your IoT controller.

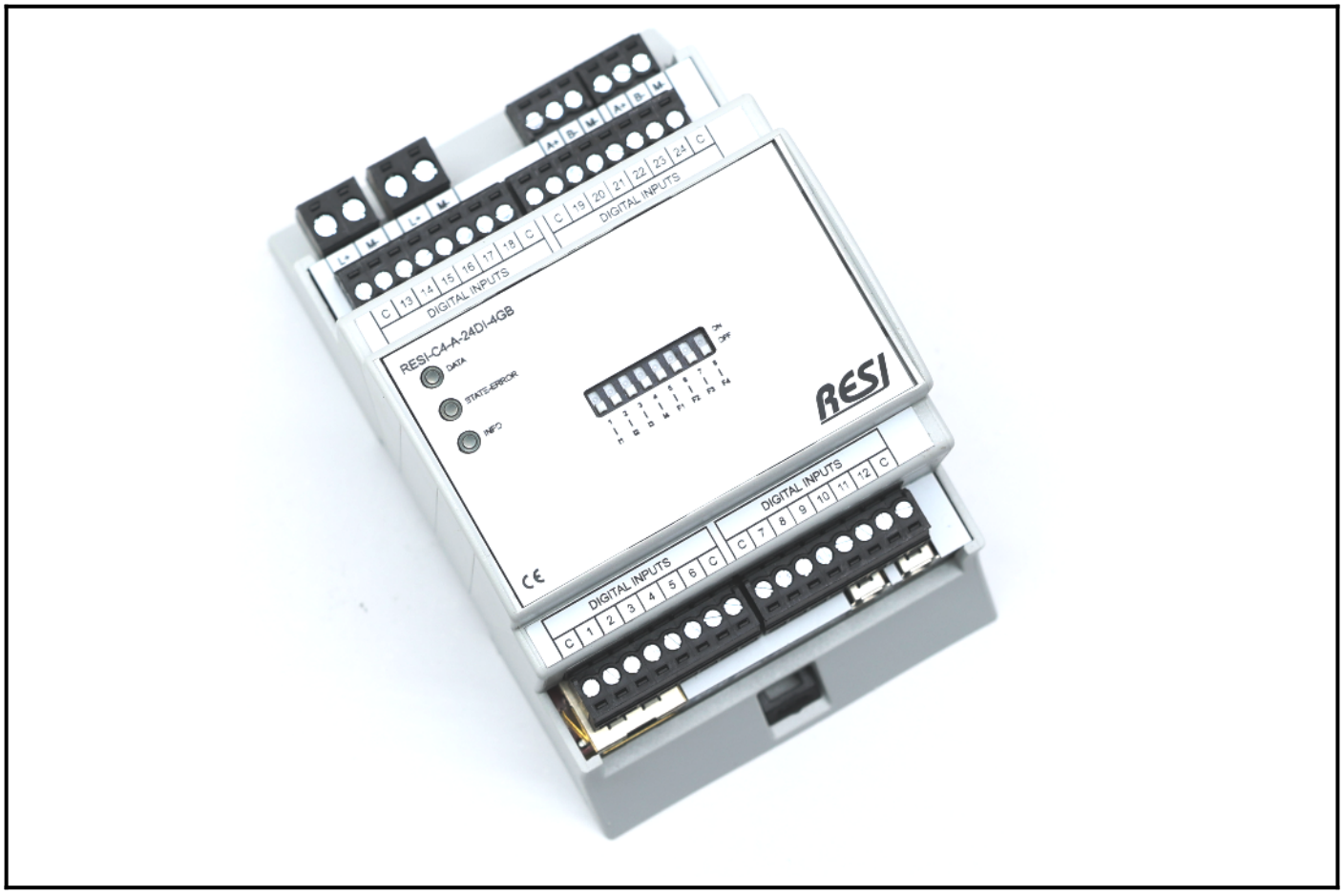


Figure: Our RESI-C4-A-24DI IoT controller with 24 digital inputs

8.3.1.1Technical specification

The digital inputs meets the following technical specification

DIGITAL INPUTS	
Sampling rate	As fast as possible
	Internal software filter ~35-80ms

DC rating	
Input voltage range	12-48V= +/-10%
Input current	per channel
	approx. 0,8mA@12V=
	approx. 1.5mA@20V=
	approx. 1.8mA@24V=
	approx. 2.5mA@32V=
	approx. 4.0mA@48V=
Input power consumption	max. 0.3W/channel
Logic levels	0: <3.8V=
	1: >4.7V=

Terminal type	RM3.5
Galvanic insulation	No, ground of digital inputs is wired to ground of IoT controller

8.3.1.2 Additional terminals or functionalities

Depending on the module the digital inputs are grouped in 6 or 16 inputs on one terminal block

DIGITAL INPUTS		
Input groups	Terminal type:	RM3.5
	C:	Common ground: wired to system ground
	1..n:	Digital input 1-n
		0=open or connected to ground
		1=DC voltage between 12 and 48V=
Pin layout	6 digital inputs for 12-48Vdc signals	
	One 8 pin plug-in terminal block	
	Pin 1:	C: Common ground
	Pin 2:	1: Digital input #1
	...	
	Pin 7:	6: Digital input #6
	Pin 8:	C: Common ground
or	16 digital inputs for 12-48Vdc signals	
	One 18 pin plug-in terminal block	
	Pin 1:	C: Common ground
	Pin 2:	1: Digital input #1
	...	
	Pin 17:	16: Digital input #16
	Pin 18:	C: Common ground
INFO	If at least one of the digital inputs is activated (ON), this LED is ON.	
	If none of the digital inputs are activated (OFF), this LED is OFF.	

8.3.1.3 Cabling of the digital inputs

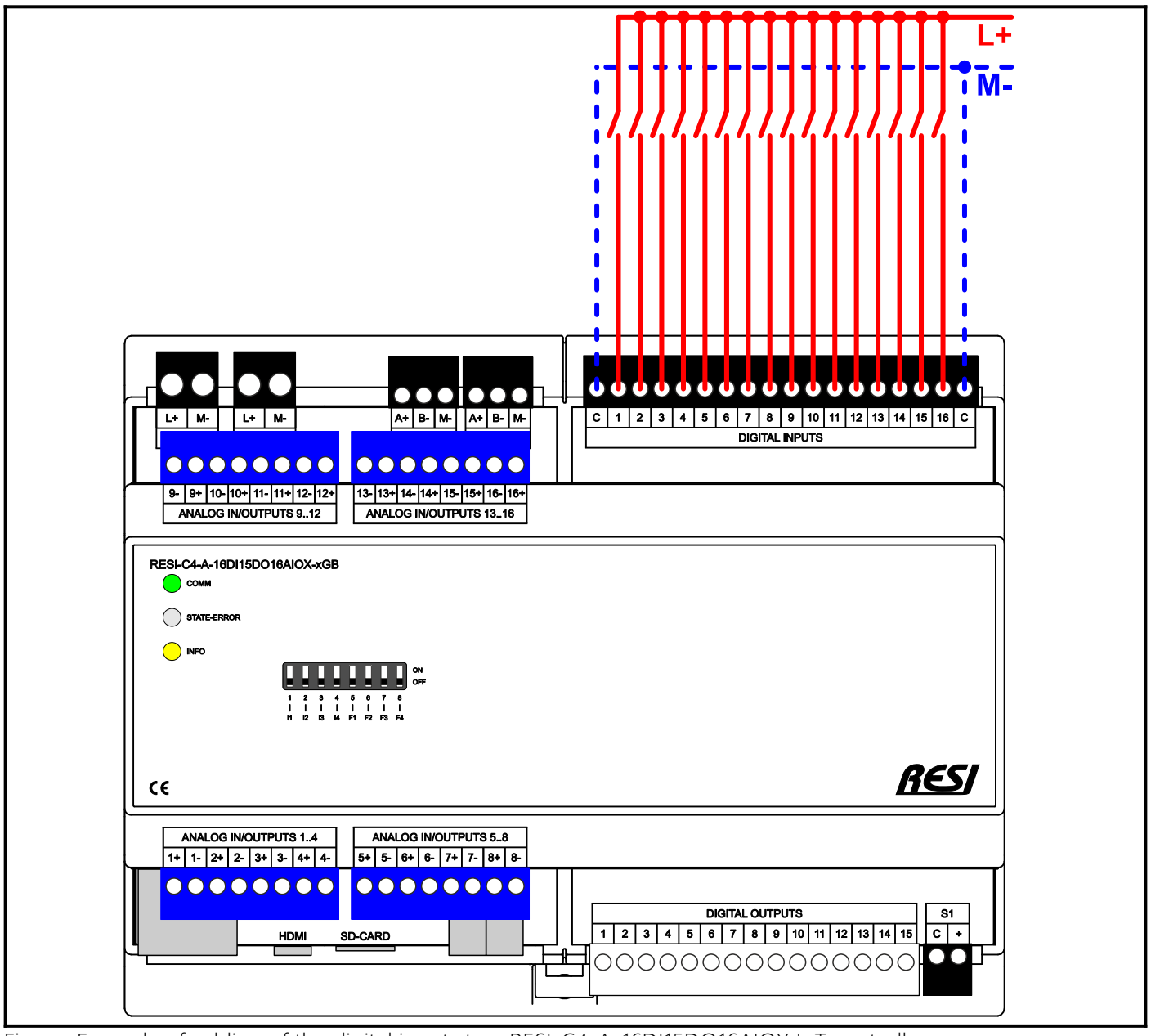


Figure: Example of cabling of the digital inputs to a RESI-C4-A-16DI15DO16AIOX IoT controller

8.3.1.4 Using the digital inputs with ASCII+MODBUS

8.3.1.4.1 Digital input filter

Our digital inputs are read internally as fast as possible. But we filter the digital inputs with a software filter based on 5ms filter time. The digital filter looks every 5ms at the last 16 samplings. If more than 7 samples have the state ON, the actual input status will be ON. So some glitches on the digital inputs are filtered and will have no effect. So the response time is around 35-80ms depending how many glitches your signal has.

8.3.1.4.2 Current status of digital inputs

In ASCII you can read the current status of the digital inputs with the commands GDIS or GDIx:

ASCII COMMANDS				
DIGITAL INPUTS				
GET DIGITAL INPUTS	ASCII READ COMMAND	#GDIS<CR> Result: #GDIS:<DISADec>,<DISBDec>,<DISAHex>,<DISBHex> <CR>	ASCII	
	TX	#255,GDIS<CR>		
	RX	#255,GDIS:0,2147483648,0x0,0x80000000<CR>		
		Actual status of digital inputs:		
		DI1-32:0000.0000.0000.0000.0000.0000.0000.0000		
		DI33-64:1000.0000.0000.0000.0000.0000.0000.0000		
<DISADec>,<DISAHex>:DI1-32 <DISBDec>,<DISBHex>:DI33-64				
Returns the actual state of all digital inputs as decimal number and as hexadecimal number. DISADec, DISAHex The current state of all digital inputs: Bit 0: State of DI1 (=0:OFF, =1:ON) Bit 1: State of DI2 (=0:OFF, =1:ON) Bit 2: State of DI3 (=0:OFF, =1:ON) ... Bit 29: State of DI30 (=0:OFF, =1:ON) Bit 30: State of DI31 (=0:OFF, =1:ON) Bit 31: State of DI32 (=0:OFF, =1:ON)				
DISBDec, DISBHex The current state of all digital inputs: Bit 0: State of DI33 (=0:OFF, =1:ON) Bit 1: State of DI34 (=0:OFF, =1:ON) Bit 2: State of DI35 (=0:OFF, =1:ON) ... Bit 29: State of DI62 (=0:OFF, =1:ON) Bit 30: State of DI63 (=0:OFF, =1:ON) Bit 31: State of DI64 (=0:OFF, =1:ON)				
GET DIGITAL INPUT DIx	ASCII READ COMMAND	#GDI<DINR><CR> Result: #GDI<DINR>:<DIxDec>,<DIxHex><CR>	ASCII	
	DINR	1		
	TX	#255,GDI1<CR>		
	RX	#255,GDI1:0,0x0<CR>		
		Actual status of digital input DI1:0=OFF		
<DINR>: 1=DI1..64=DI64				

In MODBUS you have many coils and registers which will show the actual digital input state:
Here are registers for coils or inputs (Every input as one bit):

Register NAME Command NAME	MODBUS Register ASCII Command	Register VALUE ASCII Command	NEW REAL VALUE	NEW VALUE	DATA TYPE	DO WRITE
STATUS DIGITAL INPUTS						
DI1	1x00001 2x00001 I:0	????			BIT R/O	
		Actual state of DI1:0=OFF				
Current state of the digital input DIx =0:DI is OFF, =1:DI is ON						
DI2	1x00002 2x00002 I:1	????			BIT R/O	
		Actual state of DI2:0=OFF				
DI3	1x00003 2x00003 I:2	????			BIT R/O	
		Actual state of DI3:0=OFF				

The same readout can be done by holding or input registers:

STATUS DIGITAL INPUTS						
DI1	3x00001 4x00001 I:0	0,0x0000 B:00 00			UINT16 R/O	
		Actual state of DI1:0=OFF				
Current state of the digital input DIx =0:DI is OFF, =1:DI is ON						
DI2	3x00002 4x00002 I:1	0,0x0000 B:00 00			UINT16 R/O	
		Actual state of DI2:0=OFF				

But you can also read all digital inputs together:

STATUS OF ALL DIS DI1..DI16	3x10002 4x10002 I:10001	0,0x0000 B:00 00			UINT16 R/O	
		Actual state of DI1:0=OFF				
		Actual state of DI2:0=OFF				
		Actual state of DI3:0=OFF				
		Actual state of DI4:0=OFF				
		Actual state of DI5:0=OFF				
		Actual state of DI6:0=OFF				
		Actual state of DI7:0=OFF				
		Actual state of DI8:0=OFF				
		Actual state of DI9:0=OFF				
		Actual state of DI10:0=OFF				
		Actual state of DI11:0=OFF				
		Actual state of DI12:0=OFF				
		Actual state of DI13:0=OFF				
		Actual state of DI14:0=OFF				
		Actual state of DI15:0=OFF				
		Actual state of DI16:0=OFF				
Actual state of all digital inputs DI1..DI12 Bit 0: =0:DI1 is OFF, =1:DI1 is ON Bit 1: =0:DI2 is OFF, =1:DI2 is ON ... Bit 14: =0:DI15 is OFF, =1:DI15 is ON Bit 15: =0:DI16 is OFF, =1:DI16 is ON						
STATUS OF ALL DIS DI17..DI32	3x10003 4x10003 I:10002	0,0x0000 B:00 00			UINT16 R/O	
		Actual state of DI17:0=OFF				
		Actual state of DI18:0=OFF				
		Actual state of DI19:0=OFF				
		Actual state of DI20:0=OFF				
		Actual state of DI21:0=OFF				
		Actual state of DI22:0=OFF				
		Actual state of DI23:0=OFF				
		Actual state of DI24:0=OFF				
		Actual state of DI25:0=OFF				
		Actual state of DI26:0=OFF				
		Actual state of DI27:0=OFF				
		Actual state of DI28:0=OFF				
		Actual state of DI29:0=OFF				
		Actual state of DI30:0=OFF				
		Actual state of DI31:0=OFF				
		Actual state of DI32:0=OFF				
Actual state of all digital inputs DI1..DI12 Bit 0: =0:DI17 is OFF, =1:DI17 is ON Bit 1: =0:DI18 is OFF, =1:DI18 is ON ... Bit 14: =0:DI31 is OFF, =1:DI31 is ON Bit 15: =0:DI32 is OFF, =1:DI32 is ON						

8.3.1.4.3 Change & event counter for inputs

The firmware in the co-processor detects the following events for every digital input:

- Rising edge
- Falling edge
- Short keypress
- Long keypress start
- Long keypress end

To detect very fast in your software if anything on the inputs has changed since your last poll use this ASCII command:

GET ALL CHANGES	ASCII READ COMMAND	#GAC<CR> Result: #GAC:<ChangesDec>,<ChangesHex><CR>	ASCII	
	TX	#255,GAC<CR>		
	RX	#255,GAC:8,0x8<CR>		
		Actual change counter:8		
Returns the counter for changes on all digital inputs. As soon as the module detects a short keypress or long key press or long key release event, this counter is incremented by 1. If this values has changed sience the last polling request, the host knows, that at least one digital input has changed its state.				

On MODBUS read this register:

HAS DIS CHANGED	3x10001 4x10001 I:10000	32,0x0020 B:00 20			UINT16 R/O	
		32 event(s)				
As soon as the module registrates an event on one of the available digital inputs, this global event counter is incremented by 1. Possible events are: Detection of a short keypress Detection of the start of a long keypress Detection of the end of a long keypress						

Its an event counter starting with 0 after power on.
So this value can be incremented not only by 1 for the next readout. But in your software you can save the last readout value. If the new readout value is different to the saved one, you know, that something has happened on the digital inputs.

But you can also readout the changes for every digital input separately:
In ASCII use this command to read blocks of 16 inputs:

CHANGE ALL DIS PART x	ASCII READ COMMAND	#CADISP<PART><CR> Result: #CADISP<PART>:<ChangeDInDec>,...,<ChangeDIn+15Dec>, <ChangeDInHex>,...,<ChangeDIn+15Hex><CR>	ASCII	
	PART	4		
	TX	#255,CADISP4<CR>		
	RX	#255,CADISP4:0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,2,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x3,0x2<CR>		
		Actual counter for changes on DI49:0		
		Actual counter for changes on DI50:0		
		Actual counter for changes on DI51:0		
		Actual counter for changes on DI52:0		
		Actual counter for changes on DI53:0		
		Actual counter for changes on DI54:0		
		Actual counter for changes on DI55:0		
		Actual counter for changes on DI56:0		
		Actual counter for changes on DI57:0		
		Actual counter for changes on DI58:0		
		Actual counter for changes on DI59:0		
		Actual counter for changes on DI60:0		
		Actual counter for changes on DI61:0		
		Actual counter for changes on DI62:0		
		Actual counter for changes on DI63:3		
		Actual counter for changes on DI64:2		
<PART>: 1..4, 1=DI1-DI16, 2=DI17-DI32, 3=DI33-DI48, 4=DI49-DI64				
Returns for each digital input the counter for changes. As soon as the module detects a signal change on a digital input, the change counter for the affected digital input is incremented by 1. A signal change can be: Detection of a short keypress Detection of the start of a long keypress Detection of a release of a long keypress The parameter <PART> defines the part of the digital inputs. The command returns maximal 16 digital inputs.				

or for only one digital input:

CHANGE DIx	ASCII READ COMMAND	#CDI<DINR><CR> Result: #CDI<DINR>:<ChangesDec>,<ChangesHex><CR>	ASCII	
	DINR	1		
	TX	#255,CDI1<CR>		
	RX	#255,CDI1:1,0x1<CR>		
		Actual counter for changes on digital input DI1:1		
<DINR>: 1=DI1..64=DI64				
Returns for digital input <DINR> the counter for signal changes. As soon as the module detects a signal change on a digital input, the change counter for the affected digital input is incremented by 1. A signal change can be: Detection of a short keypress Detection of the start of a long keypress Detection of a release of a long keypress				

On the MODBUS use this registers with one bit. On every change this bit toggles. But be aware, if you are not fast enough you can miss a change because maybe there are more than one changes on the digital inputs since your last poll.

DIGITAL INPUTS: DIGITAL INPUT HAS CHANGED IT'S STATE						
DI HAS CHANGED DI1	1x20001 2x20001 I:20000	1,0x01 B:01			BIT R/O	
If the digital input has changed this bit inverts its last state						
DI HAS CHANGED DI2	1x20002 2x20002 I:20001	1,0x01 B:01			BIT R/O	
DI HAS CHANGED DI3	1x20003 2x20003 I:20002	0,0x00 B:00			BIT R/O	
DI HAS CHANGED DI4	1x20004 2x20004 I:20003	0,0x00 B:00			BIT R/O	

Or you use a table for every digital input, which stores all events. Here you will find a counter for every type of event for one specific digital input. e.g. DI3

DIGITAL INPUTS: STATUS FOR DIGITAL INPUT DI3						
RISE DI3	3x07021 4x07021 I:7020	4,0x0004 B:00 04			UINT16 R/O	
		4 event(s)				
FALL DI3	3x07022 4x07022 I:7021	4,0x0004 B:00 04			UINT16 R/O	
		4 event(s)				
CHANGE DI3	3x07023 4x07023 I:7022	6,0x0006 B:00 06			UINT16 R/O	
		6 event(s)				
SHORT KEYPRESS DI3	3x07024 4x07024 I:7023	2,0x0002 B:00 02			UINT16 R/O	
		2 event(s)				
LONG KEYPRESS START DI3	3x07025 4x07025 I:7024	2,0x0002 B:00 02			UINT16 R/O	
		2 event(s)				
LONG KEYPRESS END DI3	3x07026 4x07026 I:7025	2,0x0002 B:00 02			UINT16 R/O	
		2 event(s)				

There is another short form table in MODBUS for every digital input with all its events for faster readout:

DIGITAL INPUTS						
STATUS DI1 A	3x05001 4x05001 I:5000	0,0x0000 B:00 00			UINT16 R/O	
		DI:0,CC:0,REC:0,FEC:0				
Status for the digital input DIx Bit 0-4: Lower 5 bits of CHANGE COUNTER Bit 5-9: Lower 5 bits of RISING EDGE COUNTER Bit 10-14: Lower 5 bits of FALLING EDGE COUNTER Bit 15: Current Status of DIx =0: DIx si OFF, =1: DIx is ON						
STATUS DI1 B	3x05002 4x05002 I:5001	0,0x0000 B:00 00			UINT16 R/O	
		DI:0,SKE:0,LKSE:0,LKEE:0				
Status for the digital input DIx Bit 0-4: Lower 5 bits of SHORT KEYPRESS EVENTS Bit 5-9: Lower 5 bits of LONG KEYPRESS START EVENTS Bit 10-14: Lower 5 bits of LONG KEYPRESS END EVENTS Bit 15: Current Status of DIx =0: DIx si OFF, =1: DIx is ON						

The same information can be read via ASCII commands:

- SHORT KEYPRESS EVENT COUNTER: SKDIx or SKADISPp
- LONG KEYPRESS START EVENT COUNTER: LKSDIx or LKSADISPp
- LONG KEYPRESS END EVENT COUNTER: LKEDIx or LKEADISPp
- RISING EDGE EVENT COUNTER: RDIx or RADISPp
- FALLING EDGE EVENT COUNTER: FDIx or FADISPp

For example here is the ASCII description for short keypress event:

SHORT KEY ALL DIS PART x	ASCII READ COMMAND	#SKADISP<PART><CR> Result: #SKADISP<PART>:<ShortKeyDInDec>,...,<ShortKeyDIn+15Dec>, <ShortKeyDInHex>,...,<ShortKeyDIn+15Hex><CR>	ASCII	
	PART	4		
	TX	#255,SKADISP4<CR>		
	RX	#255,SKADISP4:0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x1 ,0x1<CR>		
		Actual counter for short keypress events on DI49:0		
		Actual counter for short keypress events on DI50:0		
		Actual counter for short keypress events on DI51:0		
		Actual counter for short keypress events on DI52:0		
		Actual counter for short keypress events on DI53:0		
		Actual counter for short keypress events on DI54:0		
		Actual counter for short keypress events on DI55:0		
		Actual counter for short keypress events on DI56:0		
		Actual counter for short keypress events on DI57:0		
		Actual counter for short keypress events on DI58:0		
		Actual counter for short keypress events on DI59:0		
		Actual counter for short keypress events on DI60:0		
		Actual counter for short keypress events on DI61:0		
		Actual counter for short keypress events on DI62:0		
		Actual counter for short keypress events on DI63:1		
		Actual counter for short keypress events on DI64:1		
<PART>: 1..4, 1=DI1-DI16, 2=DI17-DI32, 3=DI33-DI48, 4=DI49-DI64				
Returns for each digital input the counter for short keypress events. As soon as the module detects a short keypress on a digital input, the counter for the affected digital input is incremented by 1. The parameter <PART> defines the part of the digital inputs. The command returns maximal 16 digital inputs.				
SHORT KEY DIx	ASCII READ COMMAND	#SKDI<DINR><CR> Result: #SKDI<DINR>:<ShortKeyDec>,<ShortKeyHex><CR>	ASCII	
	DINR	1		
	TX	#255,SKDI1<CR>		
	RX	#255,SKDI1:1,0x1<CR>		
		Actual counter for short keypress events on digital input DI1:1		
<DINR>: 1=DI1..64=DI64				

With the ASCII command RESET COUNTERS you can set all counters to 0

RESET COUNTERS	ASCII WRITE COMMAND	#RC<CR> Result: #OK<CR>	ASCII	NO
	TX	#255,RC<CR>		
	RX	N/A		
Resets all internal counters for digital inputs and events on this digital inputs to 0.				

The same command on MODBUS for coil:

DIGITAL INPUTS: RESET						
RESET COUNTERS	1x10000 2x10000 I:9999	????		1:PERFORM RESET	BIT R/W	NO
If this register is written to 1, all internal edge counters and event counters are set to 0. 0 is always returned when reading.						

and for holding register:

DIGITAL INPUTS: RESET						
RESET COUNTERS	3x10000 4x10000 I:9999	0,0x0000 B:00 00		1:PERFORM RESET	UINT16 R/W	NO
If this register is written to 1, all internal edge counters and event counters are set to 0. 0 is always returned when reading.						

8.3.1.4.4 ASCII Events

The firmware in the co-processor can send the actual status of all digital inputs in case of a change on the digital inputs automatically without the need of polling the current status. But be aware, that then the co-processor sends this string whenever a change occurs. This has to be handled by your software. Also the event can be send before the ASCII answer to an ASCII request is sent back!

To activate this event feature, simply send EVTON command. After that command, the controller will send at every change of a digital input a telegram in the format:

For controller with less or equal to 32 digital inputs:
#255,EVT:DIS:<StateOfDI1-32 in Dec>,<StateOfDI1-32 in Hex>
e.g. DI1 has changed from 0 to 1:
#255,EVT:DIS:1,0x1

For controller with more than 32 digital inputs:
#255,EVT:DIS:<StateOfDI1-32 in Dec>,<StateOfDI33-64 in Dec>,<StateOfDI1-32 in Hex>,<StateOfDI33-64 in Hex>
e.g. DI33 has changed form 0 to 1
#255,EVT.DIS:0,1,0x0,0x1

To switch this events off, send EVTOFF command

DIGITAL INPUTS EVENTs				
EVENTS ON	ASCII WRITE COMMAND	#EVTON<CR> Result: #OK<CR>	ASCII	NO
	TX	#255,EVTON<CR>		
	RX	#1,OK<CR>		
Activates event sending of changes on digital inputs Whenever a change is detected on the digital inputs, the IO module sends immediately #<BusAdr>,EVT:DIS:<AllDISasDec>,<AllDISasHex> <CR>				
EVENTS OFF	ASCII WRITE COMMAND	#EVTOFF<CR> Result: #OK<CR>	ASCII	NO
	TX	#255,EVTOFF<CR>		
	RX	#1,OK<CR>		
Deactivates event sending of changes on digital inputs Whenever a change is detected on the digital inputs, the IO module sends immediately #<BusAdr>,EVT:DIS:<AllDISasDec>,<AllDISasHex> <CR>				

8.3.2 Digital outputs DC ≦30V=

This output type supports DC semiconductor outputs for maximum 30Vdc output voltage. It drives every output with a maximum current of 700mA. Also the outputs are organized in groups of 6, 12 or 15 digital outputs with own output power supply. Every output group is limited to a maximum output current of 1.8A and can have its own power supply. But the grounds of all power supplies are internally connected with the M- of the IoT controller. Extensive diagnostic features are available for every output: Thermal overload, over-current, shortcut to power supply, open wire.

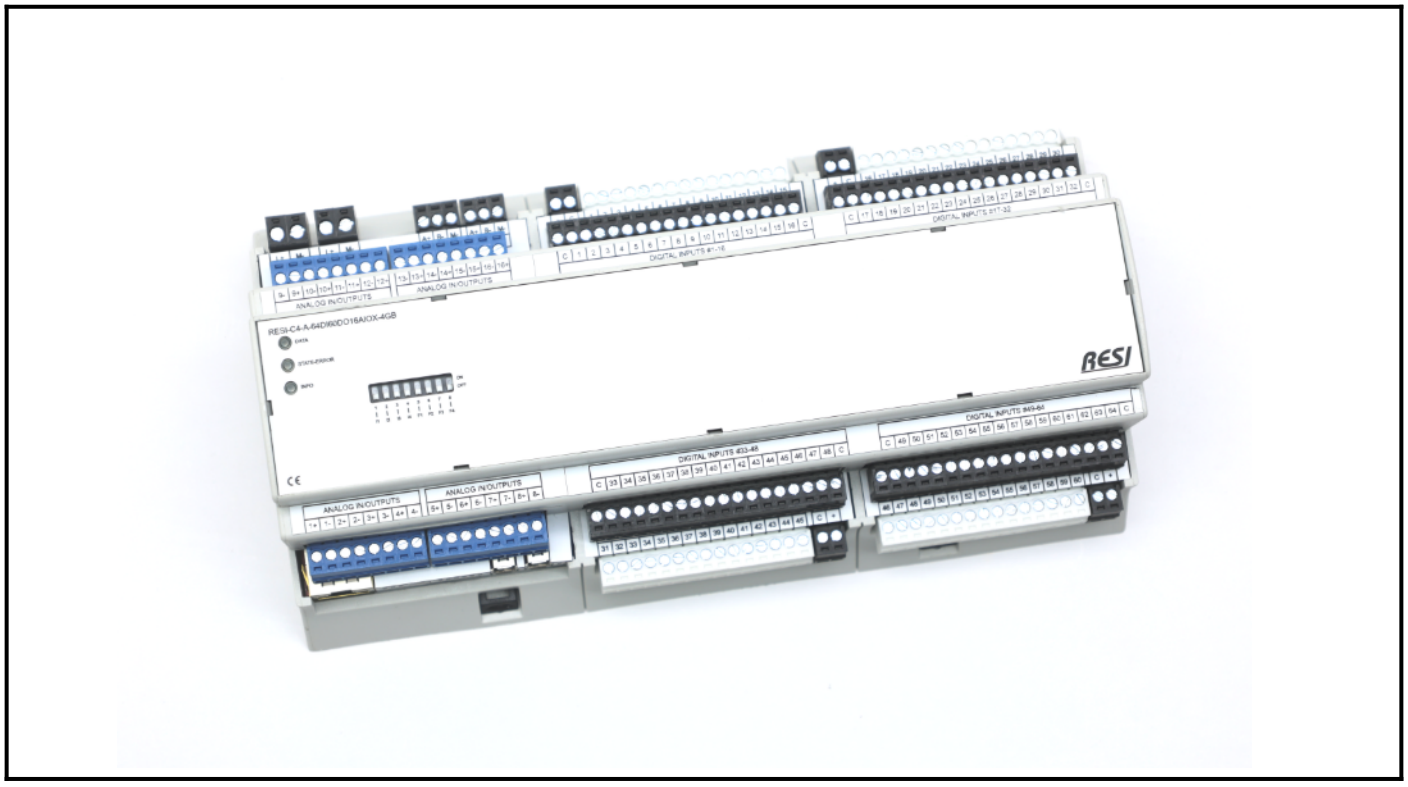


Figure: Our RESI-C4-A-64DI60DO16AIOX IoT controller with 60 digital outputs

8.3.2.1 Technical specification

The digital outputs meets the following technical specification

DIGITAL OUTPUTS	
Update rate	As fast as possible
DC rating	
Output voltage range	10-36V= +/-10%, typical 24V=
Output current	max. 700mA
Diagnostic	Loss of power supply
	Thermal overheating
	Overload
	Over-current
	Open wire in state ON and OFF
	Shortcut to power supply in state OFF
Terminal type	RM3.5
Galvanic insulation	No, the digital output groups are internally tied to ground of the IoT controller

8.3.2.2 Additional terminals or functionalities

Depending on the module the digital outputs are grouped in 6, 12 or 15 outputs on one terminal block

DIGITAL OUTPUTS		
Output groups	Terminal type:	RM3.5
	Sx:C:	Common ground: wired to system ground
	Sx:+:	Power supply input max. 30Vdc
	1..n:	Digital output 1-n
		0=output is open
		1=output is closed and delivers the voltage
		of the Sx:+ terminal
Pin layout	power supply of output group	
	Pin 1:	Sx:C: Common ground
	Pin 2:	Sx:+: Power supply 10-30Vdc
	6 digital outputs for DC signals max. 700mA	
	One 6 pin plug-in terminal block	
	Pin 1:	1: Digital output #1
	...	
	Pin 6:	6: Digital output #6
or	15 digital outputs for DC signals max. 700mA	
	One 15 pin plug-in terminal block	
	Pin 1:	1: Digital output #1
	...	
	Pin 15:	15: Digital output #15
INFO	If at least one of the digital outputs is activated (ON), this LED is ON.	
	If none of the digital outputs are activated (OFF), this LED is OFF.	

8.3.2.3 Cabling of the digital outputs

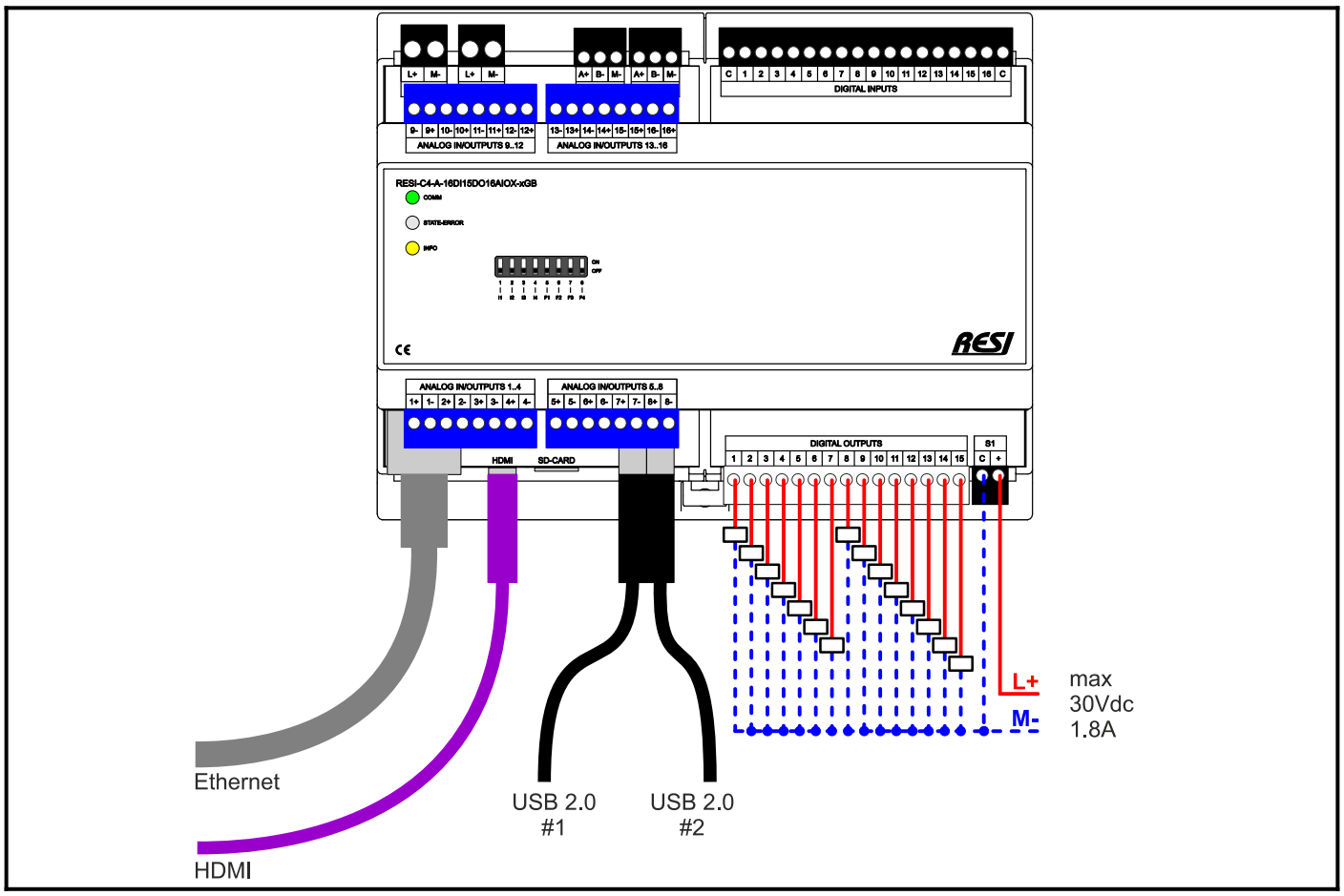


Figure: Example of cabling of the digital outputs to a RESI-C4-A-16DI15DO16AIOX IoT controller

8.3.2.4 Using the digital outputs with ASCII+MODBUS

8.3.2.4.1 Update all digital inputs & outputs

In ASCII there is a special command to update all digital outputs and read back the actual state of all digital inputs with one command:

ASCII COMMANDS				
DIGITAL OUTPUTS				
UPDATE DIGITAL INPUTS AND OUTPUTS	ASCII WRITE COMMAND	#UDIOS:<OutAllDOS><CR> Result: #UDIOS:<InAllDISDec>,<InAllDISHex><CR>	ASCII	YES
	DO1	0:OFF		
	DO2	0:OFF		
	DO3	0:OFF		
	DO4	0:OFF		
	DO5	0:OFF		
	DO6	0:OFF		
	DO7	0:OFF		
	DO8	0:OFF		
	DO9	0:OFF		
	DO10	0:OFF		
	DO11	0:OFF		
	DO12	0:OFF		
	DO13	0:OFF		
	DO14	0:OFF		
	DO15	0:OFF		
	TX	#255,UDIOS:0<CR>		
	RX	#255,UDIOS:0,0x0<CR>		
		Actual status of digital inputs:0000.0000.0000.0000		

Sets all digital outputs to the new state OutAllDOS and gives back the current status of all digital inputs InAllDIS as decimal and hexadecimal value
OutAllDOS: The new state for all digital outputs
Bit 0: State of DO1 (=0:OFF, =1:ON)
Bit 1: State of DO2 (=0:OFF, =1:ON)
Bit 2: State of DO3 (=0:OFF, =1:ON)
...
Bit 12: State of DO13 (=0:OFF, =1:ON)
Bit 13: State of DO14 (=0:OFF, =1:ON)
Bit 14: State of DO15 (=0:OFF, =1:ON)

InAllDIS: The current state for all digital inputs
Bit 0: State of DI1 (=0:OFF, =1:ON)
Bit 1: State of DI2 (=0:OFF, =1:ON)
Bit 2: State of DI3 (=0:OFF, =1:ON)
...
Bit 13: State of DI14 (=0:OFF, =1:ON)
Bit 14: State of DI15 (=0:OFF, =1:ON)
Bit 15: State of DI16 (=0:OFF, =1:ON)

8.3.2.4.2 Current status of digital outputs

In ASCII you can read the current status of the digital outputs with the commands GDOS or GDOx:

GET DIGITAL OUTPUTS	ASCII READ COMMAND	#GDOS<CR> Result: #GDOS:<DOSDec>,<DOSHex><CR>	ASCII	
	TX	#255,GDOS<CR>		
	RX	#255,GDOS:0,0x0<CR>		
		Actual status of digital outputs:000.0000.0000.0000		

Returns the actual state of the digital outputs as decimal number and as hexadecimal number.
DOSDec, DOSHex
The current state of the digital outputs:
Bit 0: State of DO1 (=0:OFF, =1:ON)
Bit 1: State of DO2 (=0:OFF, =1:ON)
Bit 2: State of DO3 (=0:OFF, =1:ON)
...
Bit 12: State of DO13 (=0:OFF, =1:ON)
Bit 13: State of DO14 (=0:OFF, =1:ON)
Bit 14: State of DO15 (=0:OFF, =1:ON)

GET DIGITAL OUTPUT DOx	ASCII READ COMMAND	#GDO<DONR><CR> Result: #GDO<DONR>:<DOxDec>,<DOxHex><CR>	ASCII	
	DONR	2		
	TX	#255,GDO2<CR>		
	RX	#255,GDO2:0,0x0<CR>		
		Actual status of digital output DO2:0=OFF		

Returns the actual state of the digital output DOx as decimal number and as hexadecimal number.
DOxDec, DOxHex
The current state of the digital output DOx:
=0: relay output is OFF
=1: relay output is ON

For writing an new state to a digital output in ASCII use the command SDOS or SDOx:

SET DIGITAL OUTPUTS	ASCII WRITE COMMAND	#SDOS:<OutAllDOS><CR> Result: #OK<CR>	ASCII	YES
	DO1	0:OFF		
	DO2	0:OFF		
	DO3	0:OFF		
	DO4	0:OFF		
	DO5	0:OFF		
	DO6	0:OFF		
	DO7	0:OFF		
	DO8	0:OFF		
	DO9	0:OFF		
	DO10	0:OFF		
	DO11	0:OFF		
	DO12	0:OFF		
	DO13	0:OFF		
	DO14	0:OFF		
	DO15	0:OFF		
	TX	#255,SDOS:0<CR>		
	RX	#255,OK<CR>		
Sets all digital outputs to the new state OutAllDOS The new state for all digital outputs Bit 0: State of DO1 (=0:OFF, =1:ON) Bit 1: State of DO2 (=0:OFF, =1:ON) Bit 2: State of DO3 (=0:OFF, =1:ON) ... Bit 12: State of DO13 (=0:OFF, =1:ON) Bit 13: State of DO14 (=0:OFF, =1:ON) Bit 14: State of DO15 (=0:OFF, =1:ON)				
SET DIGITAL OUTPUT DOx	ASCII WRITE COMMAND	#SDO<DONR>:<Out><CR> Result: #OK<CR>	ASCII	NO
	DONR	2		
	DOx	0:OFF		
	TX	#255,SDO2:0<CR>		
	RX	N/A		
<DONR>: 1=DO1..15=DO15				

In MODBUS you have many coils and registers which will show the actual digital output state or with which you can set a new output state:

Here are tables for coils (every output as one bit):

STATUS DIGITAL OUTPUTS						
DO1	1x00017 2x00017 I:16	????		1	BIT R/W	NO
		Actual state of DO1:0=OFF		ENTER NEW STATE (0 or 1)		
Current state of the digital output DOx =0:DO is OFF, =1:DO is ON Writing on this register changes the state of the digital output						
DO2	1x00018 2x00018 I:17	????		0	BIT R/W	NO
		Actual state of DO2:0=OFF		ENTER NEW STATE (0 or 1)		

The same reading and writing can be done by holding registers:

STATUS DIGITAL OUTPUTS						
DO1	3x00017 4x00017 I:16	0,0x0000 B:00 00		1	UINT16 R/W	NO
		Actual state of DO1:0=OFF		ENTER NEW STATE (0 or 1)		
Current state of the digital output DOx =0:DO is OFF, =1:DO is ON Writing on this register changes the state of the digital output						
DO2	3x00018 4x00018 I:17	0,0x0000 B:00 00		0	UINT16 R/W	NO
		Actual state of DO2:0=OFF		ENTER NEW STATE (0 or 1)		

But you can also read and write to digital output groups in holding register:

STATUS OF ALL DOS DO1-DO15	3x10003 4x10003 I:10002	0,0x0000 B:00 00		0x7FFF	UINT16 R/W	NO
		Actual state of DO1:0=OFF		1		
		Actual state of DO2:0=OFF		1		
		Actual state of DO3:0=OFF		1		
		Actual state of DO4:0=OFF		1		
		Actual state of DO5:0=OFF		1		
		Actual state of DO6:0=OFF		1		
		Actual state of DO7:0=OFF		1		
		Actual state of DO8:0=OFF		1		
		Actual state of DO9:0=OFF		1		
		Actual state of DO10:0=OFF		1		
		Actual state of DO11:0=OFF		1		
		Actual state of DO12:0=OFF		1		
		Actual state of DO13:0=OFF		1		
		Actual state of DO14:0=OFF		1		
		Actual state of DO15:0=OFF		1		
Actual state of all digital outputs Bit 0: =0:DO1 is OFF, =1:DO1 is ON Bit 1: =0:DO2 is OFF, =1:DO2 is ON ... Bit 13: =0:DO14 is OFF, =1:DO14 is ON Bit 14: =0:DO15 is OFF, =1:DO15 is ON Write on this register sets all digital outputs to a new state						

8.3.2.4.3 Pulsing the digital outputs

With a special ASCII command you can initiate a one time pulse on a specific output. You can read back the remaining duration of the current output pulse.

DIGITAL OUTPUTS: PULSE OUTPUT				
PULSE DOx	ASCII WRITE COMMAND	#PDO<DONR>:<Time><CR> Result: #OK<CR>	ASCII	YES
	DONR	2		
	TIME	200		
	TX	#255,PDO2:200<CR>		
	RX	#255,OK<CR>		
<DONR>: 1=DO1..15=DO15				
<Time>: 0..65535*100ms				
This command switches the digital output DOx on for the pulse duration <PulseTimeIn100ms>*100ms. PulseTimeIn100ms: A duration in 100ms units. The corresponding digital output is switched on for this time period.				
GET PULSE TIMER DOx	ASCII READ COMMAND	#GPT<DONR><CR> Result: #GPT:<TimeDec>,<TimeHex><CR>	ASCII	
	DONR	2		
	TX	#255,GPT2<CR>		
	RX	#255,GPT2:19810,0x4D62<CR>		
		Actual pulse time for DO2:19,8s		
<DONR>: 1=DO1..15=DO15				
Returns the remaining timer value of the pulse for digital output DOx in ms. PulseTimeInMSDec, PulseTimeInMSHex The remaining time of the pulse in Milliseconds				

But you can also initiate this digital output pulse with this MODBUS registers:

PULSE TIME FOR DIGITAL OUTPUTS						
PULSE TIME DO1	3x20001 4x20001 I:20000	0,0x0000 B:00 00	200	20,0	UINT16 R/W	YES
Generate a pulse on digital output x in 100ms units (0,1 to 6553,5 Seconds selectable) If you write onto this register, the digital output will be switched on for the desired time in 100ms units.						
PULSE TIME DO2	3x20002 4x20002 I:20001	0,0x0000 B:00 00	300	30,0	UINT16 R/W	NO
PULSE TIME DO3	3x20003 4x20003 I:20002	0,0x0000 B:00 00	400	40,0	UINT16 R/W	NO
PULSE TIME DO4	3x20004 4x20004 I:20003	0,0x0000 B:00 00	500	50,0	UINT16 R/W	NO

The remaining time for the current pulse can be read with this registers:

PULSE STATUS FOR DIGITAL OUTPUTS						
PULSE TIMER DO1	3x21001 4x21001 I:21000	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		0,0 seconds				
Remaining time of the pulse on digital output x in Milliseconds.						
PULSE TIMER DO2	3x21003 4x21003 I:21002	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		0,0 seconds				
PULSE TIMER DO3	3x21005 4x21005 I:21004	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		0,0 seconds				

PULSE STATUS FOR DIGITAL OUTPUTS						
PULSE TIMER DO1	3x21031 4x21031 I:21030	0,0x00000000 B:00 00 00 00			UINT32R R/O	
		0,0 seconds				
Remaining time of the pulse on digital output x in Milliseconds.						
PULSE TIMER DO2	3x21033 4x21033 I:21032	0,0x00000000 B:00 00 00 00			UINT32R R/O	
		0,0 seconds				

8.3.2.4.4 Diagnostic information for digital outputs

Our digital outputs offer many commands to retrieve diagnostic information of a digital output or a chipset for a group of digital outputs.

The main diagnostic features per output are:

- Detect open wire while DO=ON
- Detect open wire while DO=OFF
- Detect shortcut to power supply (VDD) while DO=OFF
- Detect thermal overload for DOx
- Detect current limit for DOx

The main diagnostic features for a chipset are:

- Is SPI communication OK
- An overload situation was detected
- A current limit was detected
- A supply error was detected
- A communication error was detected
- Internal under voltage detected
- External power supply VDD not good detected (<17V)
- External power supply VDD warning detected (<12V)
- External power supply VDD under voltage detected (<8V)
- Thermal shutdown

8.3.2.4.4.1 General diagnostic status of every chip

First we concentrate on the general diagnostic features per chipset.
Every chipset can drive up to 8 digital outputs. So for a 30DO module, we build four chips into the module.

With the command GDOINTS you will get the status of all chips. When you want to read out only one chip use GDOINTx command. Please refer to the command list for the meaning of every bit.

DIGITAL OUTPUTS: INTERRUPT STATUS			
GET DIGITAL OUTPUTS INTERRUPT STATUS	ASCII READ COMMAND	#GDOINTS<CR> Result: #GDOINTS:<InterruptStatusDec>,<InterruptStatusHex><CR>	ASCII
	TX	#255,GDOINTS<CR>	
	RX	#255,GDOINTS:16448,0x4040<CR>	
		Actual interrupt status of all digital output groups:	
		CHIP #1:0100.0000	
		CHIP #2:0100.0000	

digital output group #1, chip #1:DO1-DO8
digital output group #2, chip #2:DO9-DO15

Returns the actual interrupt state of all output groups as decimal number and as hexadecimal number.
InterruptStatusDec,InterruptStatusHex: The current interrupt state of digital output group 1-4 (CHIP1-4):
For each chip 8 bits are used: CHIP#1:Bits 0-7, CHIP#2:Bits 8-15

Bit 0: Overload detected (0=OK,1=FAULT)
Bit 1: Current limit detected(0=OK,1=FAULT)
Bit 2: Open wire while OFF detected (0=OK,1=FAULT)
Bit 3: Open wire while ON detected (0=OK,1=FAULT)
Bit 4: Short to VDD while ON detected (0=OK,1=FAULT)
Bit 5: Thermal error detected-shutdown (0=OK,1=FAULT)
Bit 6: Supply error detected (0=OK,1=FAULT)
Bit 7: Communication error detected (0=OK,1=FAULT)

GET DIGITAL OUTPUT GROUPx INTERRUPT STATUS	ASCII READ COMMAND	#GDOINT<DOGRP><CR> Result: #GDOINT<DOGRP>:<InterruptStatusDec>,<InterruptStatusHex><CR>	ASCII
	DOGRP	2	
	TX	#255,GDOINT2<CR>	
	RX	#255,GDOINT2:64,0x40<CR>	
		Actual interrupt status of digital output group 2:0100.0000	

<DOGRP>: 1=CHIP1..2=CHIP2

digital output group #1, chip #1:DO1-DO8
digital output group #2, chip #2:DO9-DO15

Returns the actual interrupt state of the digital output group DOGRP as decimal number and as hexadecimal number.
InterruptStatusDec, InterruptStatusHex
Bit 0:Overload detected (0=OK,1=FAULT)
Bit 1:Current limit detected(0=OK,1=FAULT)
Bit 2:Open wire while OFF detected (0=OK,1=FAULT)
Bit 3:Open wire while ON detected (0=OK,1=FAULT)
Bit 4:Short to VDD while ON detected (0=OK,1=FAULT)
Bit 5:Thermal error detected-shutdown (0=OK,1=FAULT)
Bit 6:Supply error detected (0=OK,1=FAULT)
Bit 7:Communication error detected (0=OK,1=FAULT)

The same information you can retrieve from the MODBUS. You can read coils or holding registers to retrieve the information:

DIGITAL OUTPUTS: INTERRUPT STATUS					
CHIP #1:DO1-DO8					
INTERRUPT STATUS BIT 0	1x00168 2x00168 I:167	????			BIT R/O
		BIT 0:Overload detected:0=OK			
INTERRUPT STATUS BIT 1	1x00169 2x00169 I:168	????			BIT R/O
		BIT 1:Current limit detected:0=OK			
INTERRUPT STATUS BIT 2	1x00170 2x00170 I:169	????			BIT R/O
		BIT 2:Open wire while OFF detected:0=OK			
INTERRUPT STATUS BIT 3	1x00171 2x00171 I:170	????			BIT R/O
		BIT 3:Open wire while ON detected:0=OK			

DIGITAL OUTPUTS: INTERRUPT STATUS					
CHIP #1:DO1-DO8					
INTERRUPT STATUS BIT 0	3x00168 4x00168 I:167	0,0x0000 B:00 00			UINT16 R/O
		BIT 0:Overload detected:0=OK			
INTERRUPT STATUS BIT 1	3x00169 4x00169 I:168	0,0x0000 B:00 00			UINT16 R/O
		BIT 1:Current limit detected:0=OK			
INTERRUPT STATUS BIT 2	3x00170 4x00170 I:169	0,0x0000 B:00 00			UINT16 R/O
		BIT 2:Open wire while OFF detected:0=OK			

Also you can read a holding register with compact information:

CHIP.#7:DO46-DO53						
CHIP.#8:DO54-DO60						
INTERRUPT STATUS FOR CHIP#7+#8	3x10049 4x10049 I:10048	16448,0x4040 B:40 40			UINT16 R/O	
		BIT 0:CHIP#3:Overload detected:0=OK				
		BIT 1:CHIP#3:Current limit detected:0=OK				
		BIT 2:CHIP#3:Open wire while OFF detected:0=OK				
		BIT 3:CHIP#3:Open wire while ON detected:0=OK				
		BIT 4:CHIP#3:Shortcut to VDD detected:0=OK				
		BIT 5:CHIP#3:Thermal shutdown:0=OK				
		BIT 6:CHIP#3:Supply error detected:1=FAULT				
		BIT 7:CHIP#3:Communication error detected:0=OK				
		BIT 8:CHIP#4:Overload detected:0=OK				
		BIT 9:CHIP#4:Current limit detected:0=OK				
		BIT 10:CHIP#4:Open wire while OFF detected:0=OK				
		BIT 11:CHIP#4:Open wire while ON detected:0=OK				
		BIT 12:CHIP#4:Shortcut to VDD detected:0=OK				
		BIT 13:CHIP#4:Thermal shutdown:0=OK				
		BIT 14:CHIP#4:Supply error detected:1=FAULT				
		BIT 15:CHIP#4:Communication error detected:0=OK				
The interrupt state for the output group. Each bit stands for a different error =0:No fault, =1:Fault						

The command GDOERRS will retrieve the error status of all chips:

DIGITAL OUTPUTS: GLOBAL ERRORS				
GET DIGITAL OUTPUTS GLOBAL ERRORS	ASCII READ COMMAND	#GDOERRS<CR> Result: #GDOERRS:<GlobalErrorsADec>,<GlobalErrorsBDec>, <GlobalErrorsAHex>,<GlobalErrorsBHex><CR>	ASCII	
	TX	#255,GDOERRS<CR>		
	RX	#255,GDOERRS:471604252,471597056,0x1C1C1C1C,0x1C1C0000<CR>		
		Actual global errors of all digital output groups:		
		CHIP #1:0001.1100		
		CHIP #2:0001.1100		
		CHIP #3:0001.1100		
		CHIP #4:0001.1100		
		CHIP #5:0000.0000		
		CHIP #6:0000.0000		
		CHIP #7:0001.1100		
		CHIP #8:0001.1100		
digital output group #1, chip #1:DO1-DO7 digital output group #2, chip #2:DO8-DO15 digital output group #3, chip #3:DO16-DO22 digital output group #4, chip #4:DO23-DO30 digital output group #5, chip #5:DO31-DO38 digital output group #6, chip #6:DO39-DO45 digital output group #7, chip #7:DO46-DO53 digital output group #8, chip #8:DO54-DO60				
Returns the actual current global error state of all output groups as decimal number and as hexadecimal number. GlobalErrorADec,GlobalErrorAHex: The current global error state of digital output groups 1-4 (CHIP1-4): For each chip 8 bits are used: CHIP#1:Bits 0-7, CHIP#2:Bits 8-15, CHIP#3:Bits 16-23, CHIP#4:Bits 24-31 GlobalErrorBDec,GlobalErrorBHex: The current current global error state of digital output groups 5-8 (CHIP5-8): For each chip 8 bits are used: CHIP#5:Bits 0-7, CHIP#6:Bits 8-15, CHIP#7:Bits 16-23, CHIP#8:Bits 24-31 Bit 0: Internal under voltage detected (0=OK,1=FAULT) Bit 1: VA under voltage detected (<2.3V) (0=OK,1=FAULT) Bit 2: VDD not good detected (<17V) (0=OK,1=FAULT) Bit 3: VDD warning detected (<12V) (0=OK,1=FAULT) Bit 4: VDD under voltage detected (<8V) (0=OK,1=FAULT) Bit 5: Thermal shutdown (0=OK,1=FAULT) Bit 6: Synchronisation error detected (0=OK,1=FAULT) Bit 7: Watchdog error detected (0=OK,1=FAULT)				

Again you can read only one chip with GDOERRx command:

GET DIGITAL OUTPUT GROUPx GLOBAL ERRORS	ASCII READ COMMAND	#GDOERR<DOGRP><CR> Result: #GDOERR<DOGRP>:<GlobalErrorsDec>,<GlobalErrorsHex><CR>	ASCII	
	DOGRP	8		
	TX	#255,GDOERR8<CR>		
	RX	#255,GDOERR8:28,0x1C<CR>		
		Actual global errors of digital output group 8:0001.1100		
<DOGRP>: 1=CHIP1..8=CHIP8				
digital output group #1, chip #1:DO1-DO7 digital output group #2, chip #2:DO8-DO15 digital output group #3, chip #3:DO16-DO22 digital output group #4, chip #4:DO23-DO30 digital output group #5, chip #5:DO31-DO38 digital output group #6, chip #6:DO39-DO45 digital output group #7, chip #7:DO46-DO53 digital output group #8, chip #8:DO54-DO60				
Returns the actual interrupt state of the digital output group DOGRP as decimal number and as hexadecimal number. InterruptStatusDec, InterruptStatusHex Bit 0:Internal under voltage detected (0=OK,1=FAULT) Bit 1:VA under voltage detected (<2.3V) (0=OK,1=FAULT) Bit 2:VDD not good detected (<17V) (0=OK,1=FAULT) Bit 3:VDD warning detected (<12V) (0=OK,1=FAULT) Bit 4:VDD under voltage detected (<8V) (0=OK,1=FAULT) Bit 5:Thermal shutdown (0=OK,1=FAULT) Bit 6:Synchronisation error detected (0=OK,1=FAULT) Bit 7:Watchdog error detected (0=OK,1=FAULT)				

Again you can read the same information via MODBUS coils or holding registers:

DIGITAL OUTPUTS: GLOBAL ERRORS						
CHIP #1:DO1-DO7						
GLOBAL ERRORS BIT 0	1x00605 2x00605 I:604	????			BIT R/O	
		BIT 0:Internal under voltage detected:0=OK				
GLOBAL ERRORS BIT 1	1x00606 2x00606 I:605	????			BIT R/O	
		BIT 1:VA under voltage detected (<2.3V):0=OK				
GLOBAL ERRORS BIT 2	1x00607 2x00607 I:606	????			BIT R/O	
		BIT 2:VDD not good detected (<17V):0=OK				
GLOBAL ERRORS BIT 3	1x00608 2x00608 I:607	????			BIT R/O	
		BIT 3:VDD warning detected (<12V):0=OK				
GLOBAL ERRORS BIT 4	1x00609 2x00609 I:608	????			BIT R/O	
		BIT 4:VDD under voltage detected (<8V):0=OK				
GLOBAL ERRORS BIT 5	1x00610 2x00610 I:609	????			BIT R/O	
		BIT 5:Thermal shutdown:0=OK				
GLOBAL ERRORS BIT 6	1x00611 2x00611 I:610	????			BIT R/O	
		BIT 6:Synchronisation error detected:0=OK				
GLOBAL ERRORS BIT 7	1x00612 2x00612 I:611	????			BIT R/O	
		BIT 7:Watchdog error detected:0=OK				
The global error state for the output group. Each bit stands for a different error =0:No fault, =1:Fault						

DIGITAL OUTPUTS: GLOBAL ERRORS						
CHIP #1:DO1-DO7						
GLOBAL ERRORS BIT 0	3x00605 4x00605 I:604	0,0x0000 B:00 00			UINT16 R/O	
		BIT 0:Internal under voltage detected:0=OK				
GLOBAL ERRORS BIT 1	3x00606 4x00606 I:605	0,0x0000 B:00 00			UINT16 R/O	
		BIT 1:VA under voltage detected (<2.3V):0=OK				
GLOBAL ERRORS BIT 2	3x00607 4x00607 I:606	0,0x0000 B:00 00			UINT16 R/O	
		BIT 2:VDD not good detected (<17V):0=OK				
GLOBAL ERRORS BIT 3	3x00608 4x00608 I:607	0,0x0000 B:00 00			UINT16 R/O	
		BIT 3:VDD warning detected (<12V):0=OK				
GLOBAL ERRORS BIT 4	3x00609 4x00609 I:608	0,0x0000 B:00 00			UINT16 R/O	
		BIT 4:VDD under voltage detected (<8V):0=OK				
GLOBAL ERRORS BIT 5	3x00610 4x00610 I:609	0,0x0000 B:00 00			UINT16 R/O	
		BIT 5:Thermal shutdown:0=OK				
GLOBAL ERRORS BIT 6	3x00611 4x00611 I:610	0,0x0000 B:00 00			UINT16 R/O	
		BIT 6:Synchronisation error detected:0=OK				
GLOBAL ERRORS BIT 7	3x00612 4x00612 I:611	0,0x0000 B:00 00			UINT16 R/O	
		BIT 7:Watchdog error detected:0=OK				
The global error state for the output group. Each bit stands for a different error =0:No fault, =1:Fault						

Or you read the compact info from holding registers:

CHIP.#3:DO16-DO22						
CHIP.#4:DO23-DO30						
GLOBAL ERRORS FOR CHIP #3+#4	3x10043 4x10043 I:10042	7196,0x1C1C B:1C 1C			UINT16 R/O	
		BIT 0:CHIP#3:Internal under voltage detected:0=OK				
		BIT 1:CHIP#3:VA under voltage detected (<2.3V):0=OK				
		BIT 2:CHIP#3:VDD not good detected (<17V):1=FAULT				
		BIT 3:CHIP#3:VDD warning detected (<12V):1=FAULT				
		BIT 4:CHIP#3:VDD under voltage detected (<8V):1=FAULT				
		BIT 5:CHIP#3:Thermal shutdown:0=OK				
		BIT 6:CHIP#3:Synchronisation error detected:0=OK				
		BIT 7:CHIP#3:Watchdog error detected:0=OK				
		BIT 8:CHIP#4:Internal under voltage detected:0=OK				
		BIT 9:CHIP#4:VA under voltage detected (<2.3V):0=OK				
		BIT 10:CHIP#4:VDD not good detected (<17V):1=FAULT				
		BIT 11:CHIP#4:VDD warning detected (<12V):1=FAULT				
		BIT 12:CHIP#4:VDD under voltage detected (<8V):1=FAULT				
		BIT 13:CHIP#4:Thermal shutdown:0=OK				
		BIT 14:CHIP#4:Synchronisation error detected:0=OK				
		BIT 15:CHIP#4:Watchdog error detected:0=OK				
The global error state for the output group. Each bit stands for a different error =0:No fault, =1:Fault						

8.3.2.4.4.2 SPI communication status of every chip

With the command GSSDOGS you can read the current status of the internal SPI communication between the ARM co-processor and the chips for the digital outputs. For every chip there is one bit in the answer.

DIGITAL OUTPUTS: SPI STATUS					
GET SPI STATUS DIGITAL OUTPUT GROUPS	ASCII READ COMMAND	#GSSDOGS<CR> Result: #GSSDOGS:<SPIDOGSDec>,<SPIDOGSHex><CR>		ASCII	
	TX	#255,GSSDOGS<CR>			
	RX	#255,GSSDOGS:0,0x0<CR>			
		Actual SPI status of digital output groups:00			
digital output group #1, chip #1:DO1-DO8 digital output group #2, chip #2:DO9-DO15					
Returns the actual SPI communication state of the corresponding output group as decimal number and as hexadecimal number. SPIDOGSDec,SPIDOGSHex The current SPI communication state of the digital output group: Bit 0: SPI communication state for digital output group #1 (=0:NO FAULT, =1:FAULT) Bit 1: SPI communication state for digital output group #2 (=0:NO FAULT, =1:FAULT)					

The command GSSDOGx will retrieve the SIP communication status of one chip.

GET SPI STATUS DIGITAL OUTPUT GROUPx	ASCII READ COMMAND	#GSSDOG<DOGRP><CR> Result: #GSSDOG<DOGRP>:<SPIDOGxDec>,<SPIDOGxHex><CR>		ASCII	
	DOGRP	8			
	TX	#255,GSSDOG8<CR>			
	RX	#255,GSSDOG8:0,0x0<CR>			
		Actual SPI status of digital output group DOG8:0=NO FAULT			
<DOGRP>: 1=CHIP1..8=CHIP8					
digital output group #1, chip #1:DO1-DO7 digital output group #2, chip #2:DO8-DO15 digital output group #3, chip #3:DO16-DO22 digital output group #4, chip #4:DO23-DO30 digital output group #5, chip #5:DO31-DO38 digital output group #6, chip #6:DO39-DO45 digital output group #7, chip #7:DO46-DO53 digital output group #8, chip #8:DO54-DO60					
Returns the actual SPI communication state of the digital output group DOGRP as decimal number and as hexadecimal number. SPIDOGxDec, SPIDOGxHex The current SPI communication state of the digital output group DOGRP: =0: SPI communication state for output group is OK (NO FAULT) =1: SPI communication state for output group is FAULT					

To check the SPI communication status with MODBUS use for coils:

SPI COMMUNICATION DIGITAL OUTPUTS					
SPI COMMUNICATION CHIP #1: DO1-DO8	1x00184 2x00184 I:183	????		BIT R/O	
		Actual SPI communication state:0=NO FAULT			
The current monitoring state of the SPI communication for the digital output group =0:No fault, =1:Fault					
SPI COMMUNICATION CHIP #2: DO9-DO15	1x00185 2x00185 I:184	????		BIT R/O	
		Actual SPI communication state:0=NO FAULT			
The current monitoring state of the SPI communication for the digital output group =0:No fault, =1:Fault					

or for holding registers:

SPI COMMUNICATION DIGITAL OUTPUTS						
SPI COMMUNICATION CHIP #1: DO1-DO7	3x00733 4x00733 I:732	0,0x0000 B:00 00			UINT16 R/O	
		Actual SPI communication state:0=NO FAULT				
The current monitoring state of the SPI communication for the digital output group =0:No fault, =1:Fault						
SPI COMMUNICATION CHIP #2: DO8-DO15	3x00734 4x00734 I:733	0,0x0000 B:00 00			UINT16 R/O	
		Actual SPI communication state:0=NO FAULT				
The current monitoring state of the SPI communication for the digital output group =0:No fault, =1:Fault						

You can read also the SPI communication within one register:

SPI COMMUNICATION DIGITAL OUTPUTS						
CHIP #1:DO1-DO8						
CHIP #2:DO9-DO15						
SPI COMMUNICATION DIGITAL OUTPUTS	#BEZUG!	0,0x0000 B:00 00			UINT16 R/O	
		Actual SPI communcation state of CHIP#1:0=OK				
		Actual SPI communcation state of CHIP#2:0=OK				
The current monitoring state of the SPI communication for the digital output group =0:No fault, =1:Fault Current SPI communication state of all digital output groups Bit x: =0:CHIP x has no fault, =1:CHIP x SPI Fault						

8.3.2.4.4.3 Diagnostic status of every digital output

The chips return for every digital output the following diagnostic status:

- Detect open wire while DO=ON
- Detect open wire while DO=OFF
- Detect shortcut to power supply (VDD) while DO=OFF
- Detect thermal overload for DOx
- Detect current limit for DOx

In ASCII you can use the following commands to read out the current diagnostic status:

- Detect open wire while DO=ON → use GDOOWFONS for all DOs or GDOOWFONx for one specific DO
- Detect open wire while DO=OFF → use GDOOWFOFFS for all DOs or GDOOWFOFFx for one specific DO
- Detect shortcut to power supply (VDD) while DO=OFF → use GDOSVDDS for all DOs or GDOSVDDx for one specific DO
- Detect thermal overload for DOx → use GDOTOS for all DOs or GDOTOx for one specific DO
- Detect current limit for DOx → use GDOCLS for all DOs or GDCLOx for one specific DO

As an example of the ASCII commands we list here the GDOTOS and GDOTOx commands. The rest of the commands you will find in the command lists of your controller.

DIGITAL OUTPUTS: THERMAL OVERLOAD DETECTION				
GET DIGITAL OUTPUTS THERMAL OVERLOAD DETECTION	ASCII READ COMMAND	#GDOTOS<CR> Result: #GDOTOS:<StatusDOSDec>,<StatusDOSHex><CR>	ASCII	
	TX	#255,GDOTOS<CR>		
	RX	#255,GDOTOS:0,0x0<CR>		
		Actual thermal overload detection status of digital outputs:		
		DO1-DO15:000.0000.0000.0000		
Returns the actual state of the thermal overload detection for all digital outputs as decimal number and as hexadecimal number. StatusDOSDec, StatusDOSHex The current detection state of the digital outputs: Bit 0: Thermal overload detected on DO1 (=0:NO, =1:YES) Bit 1: Thermal overload detected on DO2 (=0:NO, =1:YES) ... Bit 13: Thermal overload detected on DO14 (=0:NO, =1:YES) Bit 14: Thermal overload detected on DO15 (=0:NO, =1:YES)				
GET DIGITAL OUTPUT DOx THERMAL OVERLOAD DETECTION	ASCII READ COMMAND	#GDOTO<DONR><CR> Result: #GDOTO<DONR>:<StatusDOxDec>,<StatusDOxHex><CR>	ASCII	
	DONR	2		
	TX	#255,GDOTO2<CR>		
	RX	#255,GDOTO2:0,0x0<CR>		
		Thermal overload detected on DO2:0=NO		
<DONR>: 1=DO1..15=DO15				
Returns the actual state of the thermal overload detection for digital output DOx as decimal number and as hexadecimal number. StatusDOxDec, StatusDOxHex The current detection state for digital output DOx: =0: digital output is OK =1: FAULT detected on digital output				

Again on the MODBUS side, we show as an example the coils and registers for the detect open wire while DO=ON status, the other registers are found in our command lists:

DIGITAL OUTPUTS: OPEN WIRE DETECTION STATUS WHILE ON						
OPEN WIRE FAULT WHILE ON DO1	1x00077 2x00077 I:76	????			BIT R/O	
		Actual detection state of an open wire fault in state ON for DO1:0=OK				
The current detection state of an open wire in the output state ON for the digital output DOx =0:No fault, =1:Fault-open wire detected						
OPEN WIRE FAULT WHILE ON DO2	1x00078 2x00078 I:77	????			BIT R/O	
		Actual detection state of an open wire fault in state ON for DO2:0=OK				
OPEN WIRE FAULT WHILE ON DO3	1x00079 2x00079 I:78	????			BIT R/O	
		Actual detection state of an open wire fault in state ON for DO3:0=OK				
DIGITAL OUTPUTS: OPEN WIRE DETECTION STATUS WHILE ON						
OPEN WIRE FAULT WHILE ON DO1	3x00077 4x00077 I:76	0,0x0000 B:00 00			UINT16 R/O	
		Actual detection state of an open wire fault in state ON for DO1:0=OK				
The current detection state of an open wire in the output state ON for the digital output DOx =0:No fault, =1:Fault-open wire detected						
OPEN WIRE FAULT WHILE ON DO2	3x00078 4x00078 I:77	0,0x0000 B:00 00			UINT16 R/O	
		Actual detection state of an open wire fault in state ON for DO2:0=OK				
OPEN WIRE FAULT WHILE ON DO3	3x00079 4x00079 I:78	0,0x0000 B:00 00			UINT16 R/O	
		Actual detection state of an open wire fault in state ON for DO3:0=OK				

or you read the compact status from holding registers:

OPEN WIRE DETECTION STATE WHILE OFF DO17-DO32	3x10027 4x10027 I:10026	32512,0x7F00 B:7F 00			UINT16 R/O	
		Actual state of open wire detection while OFF for DO17:0=OFF				
		Actual state of open wire detection while OFF for DO18:0=OFF				
		Actual state of open wire detection while OFF for DO19:0=OFF				
		Actual state of open wire detection while OFF for DO20:0=OFF				
		Actual state of open wire detection while OFF for DO21:0=OFF				
		Actual state of open wire detection while OFF for DO22:0=OFF				
		Actual state of open wire detection while OFF for DO23:0=OFF				
		Actual state of open wire detection while OFF for DO24:0=OFF				
		Actual state of open wire detection while OFF for DO25:1=ON				
		Actual state of open wire detection while OFF for DO26:1=ON				
		Actual state of open wire detection while OFF for DO27:1=ON				
		Actual state of open wire detection while OFF for DO28:1=ON				
		Actual state of open wire detection while OFF for DO29:1=ON				
		Actual state of open wire detection while OFF for DO30:1=ON				
		Actual state of open wire detection while OFF for DO31:1=ON				
		Actual state of open wire detection while OFF for DO32:0=OFF				
Actual diagnostic state for open wire detection while OFF for digital output DOx Bit 0: =0:Output DO17 is OK, =1:Fault-Open wire detected on DO17 Bit 1: =0:Output DO18 is OK, =1:Fault-Open wire detected on DO18 ... Bit 14: =0:Output DO31 is OK, =1:Fault-Open wire detected on DO31 Bit 15: =0:Output DO32 is OK, =1:Fault-Open wire detected on DO32						

Similar ASCII commands and MODBUS coils & registers you will find for the other diagnostic informations in out command & register list for every product.

But before this status is updated you have to select for every individual output, which type of diagnostic you want to enable. Therefore we offer the following commands:

- Detect open wire while DO=ON → Enable this diagnostic with SDOEOWDONS or one individual output with SDOEOWDONx, check the current enable status with GDOEOWDONS or for one DO with GDOEOWDONx
- Detect open wire while DO=OFF → Enable this diagnostic with SDOEOWDOFFS or one individual output with SDOEOWDOFFx, check the current enable status with GDOEOWDOFFS or for one DO with GDOEOWDOFFx
- Detect shortcut to power supply (VDD) while DO=OFF → Enable this diagnostic with SDOESVDDS or one individual output with SDOESVDDx, check the current enable status with GDOESVDDS or for one DO with GDOESVDDx
- Detect thermal overload for DOx → Is always enabled
- Detect current limit for DOx → Is always enabled

As an example here the ASCII commands for enabling the shortcut to VDD diagnostic while DO=OFF:

DIGITAL OUTPUTS: ENABLE SHORTCUT TO VDD DETECTION WHILE OFF				
SET DIGITAL OUTPUTS ENABLE SHORTCUT TO VDD DETECTION WHILE OFF	ASCII WRITE COMMAND	#SDOESVDDS:<ShortCutDOS><CR> Result: #OK<CR>	ASCII	YES
	DO1	0:OFF		
	DO2	0:OFF		
	DO3	0:OFF		
	DO4	0:OFF		
	DO5	0:OFF		
	DO6	0:OFF		
	DO7	0:OFF		
	DO8	0:OFF		
	DO9	0:OFF		
	DO10	0:OFF		
	DO11	0:OFF		
	DO12	0:OFF		
	DO13	0:OFF		
	DO14	0:OFF		
	DO15	0:OFF		
	TX	#255,SDOESVDDS:0<CR>		
	RX	#255,OK<CR>		
Sets the shortcut to VDD detection mode for all digital outputs to the new mode ShortCutDOS. This enables the diagnostic of shortcut to VDD while the digital output is OFF. The new state for all digital outputs Bit 0: New mode for DO1 (=0:DISABLED, =1:ENABLED) Bit 1: New mode for DO2 (=0:DISABLED, =1:ENABLED) ... Bit 13: New mode for DO14 (=0:DISABLED, =1:ENABLED) Bit 14: New mode for DO15 (=0:DISABLED, =1:ENABLED)				
SET DIGITAL OUTPUT ENABLE SHORTCUT TO VDD DETECTION WHILE OFF	ASCII WRITE COMMAND	#SDOESVDD<DONR>:<ShortCutDOx><CR> Result: #OK<CR>	ASCII	YES
	DONR	2		
	DOx	0:DISABLE		
	TX	#255,SDOESVDD2:0<CR>		
	RX	#255,OK<CR>		
<DONR>: 1=DO1..15=DO15				
<ShortCutDOx>: 0=DISABLE..1=ENABLE				
Sets the shortcut to VDD mode for digital output DOx to the new mode ShortCutDOx. This enables the diagnostic of shortcut to VDD while the digital output is OFF. The new mode of the digital output DOx: =0: diagnostic mode for digital output is DISABLED =1: diagnostic mode for digital output is ENABLED				
GET DIGITAL OUTPUTS ENABLE SHORT CUT TO VDD DETECTION WHILE OFF	ASCII READ COMMAND	#GDOESVDDS<CR> Result: #GDOESDDS:<ShortCutDOSDec>,<ShortCutDOSHex><CR>	ASCII	
	TX	#255,GDOESVDDS<CR>		
	RX	#255,GDOESVDDS:0,0x0<CR>		
		Actual mode for shortcut to VDD diagnostic while OFF of digital outputs:		
		DO1-DO15:000.0000.0000.0000		
Returns the actual mode for shortcut to VDD diagnostic while digital output is OFF as decimal number and as hexadecimal number. ShortCutDOSDec, ShortCutDOSHex The current mode for shortcut diagnostic while OFF of the digital outputs: Bit 0: Open wire diagnostic mode of DO1 (=0:DISABLED, =1:ENABLED) Bit 1: Open wire diagnostic mode of DO2 (=0:DISABLED, =1:ENABLED) ... Bit 13: Open wire diagnostic mode of DO14 (=0:DISABLED, =1:ENABLED) Bit 14: Open wire diagnostic mode of DO15 (=0:DISABLED, =1:ENABLED)				
GET DIGITAL OUTPUT ENABLE SHORT CUT TO VDD DETECTION WHILE OFF	ASCII READ COMMAND	#GDOESVDD<DONR><CR> Result: #GDOESVDD<DONR>:<ShortCutDOxDec>,<ShortCutDOxHex><CR>	ASCII	
	DONR	2		
	TX	#255,GDOESVDD2<CR>		
	RX	#255,GDOESVDD2:0,0x0<CR>		
		Actual shortcut to VDD diagnostic mode while OFF of digital output DO2:0=DISABLED		
<DONR>: 1=DO1..15=DO15				
Returns the actual short cut to VDD diagnostic mode while OFF of the digital output DOx as decimal number and as hexadecimal number. ShortCutDOxDec, ShortCutDOxHex The current diagnostic mode of the digital output DOx: =0: open wire diagnostic mode for digital output is DISABLED =1: open wire diagnostic mode for digital output is ENABLED				

For the other two enable commands you will find the ASCII syntax in the command & register lists of your product.

With MODBUS you can enable the same diagnostic features with coils or registers. You can enable/disable every single DO with this registers:

ENABLE OPEN WIRE DETECTION ON DO1	1x00032 2x00032 I:31	????		1	BIT R/W	NO
		Actual setup of open wire detection for state ON of DO1:0=OFF		ENTER NEW SETUP MODE (0 or 1)		
Enables/disabled detection of an open wire in DO state ON for the digital output DOx =0:Open wire detection is OFF, =1:Open wire detection is ON Writing on this register changes the state of the open wire detection for this output						
ENABLE OPEN WIRE DETECTION ON DO2	1x00033 2x00033 I:32	????		1	BIT R/W	NO
		Actual setup of open wire detection for state ON of DO2:0=OFF		ENTER NEW SETUP MODE (0 or 1)		
ENABLE OPEN WIRE DETECTION ON DO3	1x00034 2x00034 I:33	????		1	BIT R/W	NO
		Actual setup of open wire detection for state ON of DO3:0=OFF		ENTER NEW SETUP MODE (0 or 1)		

or

ENABLE OPEN WIRE DETECTION ON DO1	3x00032 4x00032 I:31	1,0x0001 B:00 01		1	UINT16 R/W	NO
		Actual setup of open wire detection for state ON of DO1:1=ON		ENTER NEW SETUP MODE (0 or 1)		
Enables/disabled detection of an open wire in DO state ON for the digital output DOx =0:Open wire detection is OFF, =1:Open wire detection is ON Writing on this register changes the state of the open wire detection for this output						
ENABLE OPEN WIRE DETECTION ON DO2	3x00033 4x00033 I:32	1,0x0001 B:00 01		1	UINT16 R/W	NO
		Actual setup of open wire detection for state ON of DO2:1=ON		ENTER NEW SETUP MODE (0 or 1)		
ENABLE OPEN WIRE DETECTION ON DO3	3x00034 4x00034 I:33	1,0x0001 B:00 01		1	UINT16 R/W	NO
		Actual setup of open wire detection for state ON of DO3:1=ON		ENTER NEW SETUP MODE (0 or 1)		

For using one compact register use:

DIGITAL OUTPUTS:ENABLE OPEN WIRE DETECTION WHILE ON						
ENABLE OPEN WIRE DETECTION WHILE ON DO1-DO15	3x10004 4x10004 I:10003	65535,0xFFFF B:FF FF		0x7FFF	UINT16 R/W	NO
		Actual setup of open wire detection while ON for DO1:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO2:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO3:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO4:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO5:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO6:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO7:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO8:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO9:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO10:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO11:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO12:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO13:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO14:1=ENABLED		1		
		Actual setup of open wire detection while ON for DO15:1=ENABLED		1		
Actual setup state for open wire detection while ON for digital output DOx Bit 0: =0:Open wire detection for DO1 is DISABLED, =1:Open wire detection for DO1 is ENABLED Bit 1: =0:Open wire detection for DO2 is DISABLED, =1:Open wire detection for DO2 is ENABLED ... Bit 13: =0:Open wire detection for DO14 is DISABLED, =1:Open wire detection for DO14 is ENABLED Bit 14: =0:Open wire detection for DO15 is DISABLED, =1:Open wire detection for DO15 is ENABLED Write on this register sets for all digital outputs a new setup state						

For the two other diagnostic features the registers are in the command & register lists for the individual product.

8.3.2.4.4.4 Configuration of diagnostic status for init & watchdog

You can set a status for the three diagnostic features, which will be used after power on or after a communication watchdog. Please refer to INIT VALUES & COMMUNICATION WATCHDOG for IOs, how this functionality works.

Use the following commands to configure this features:

- Detect open wire while DO=ON → Enable the configuration of this diagnostic with SCDOEOWDONS, check the current configuration with GCDOEOWDONS
- Detect open wire while DO=OFF → Enable the configuration of this diagnostic with SCDOEOWDOFFS, check the current configuration with GCDOEOWDOFFS
- Detect shortcut to power supply (VDD) while DO=OFF → Enable the configuration of this diagnostic with SCDOESVDDS, check the current configuration with GCDOESVDDS
- Detect thermal overload for DOx → Is always enabled
- Detect current limit for DOx → Is always enabled

Here are the ASCII commands for the configuration of the diagnostic function detect shortcut to power supply (VDD) while DO=OFF:

DIGITAL OUTPUTS: INIT & WATCHDOG ENABLE SHORTCUT TO VDD DETECTION WHILE OFF				
SET CONFIG DIGITAL OUTPUTS ENABLE SHORTCUT TO VDD DETECTION WHILE OFF	ASCII WRITE COMMAND	#SCDOESVDDS:<ShortCutDOS><CR> Result: #OK<CR>	ASCII	YES
	DO1	1:ENABLE		
	DO2	1:ENABLE		
	DO3	1:ENABLE		
	DO4	1:ENABLE		
	DO5	1:ENABLE		
	DO6	1:ENABLE		
	DO7	1:ENABLE		
	DO8	1:ENABLE		
	DO9	1:ENABLE		
	DO10	1:ENABLE		
	DO11	1:ENABLE		
	DO12	1:ENABLE		
	DO13	1:ENABLE		
	DO14	1:ENABLE		
	DO15	1:ENABLE		
	TX	#255,SCDOESVDDS:32767<CR>		
	RX	#255,OK<CR>		
Sets the shortcut to VDD detection mode for all digital outputs to the new mode ShortCutDOS for init & watchdog. This enables the diagnostic of shortcut to VDD while the digital output is OFF. This state is used after power on and after a communication watchdog timeout, if a watchdog time is configured The new state for all digital outputs Bit 0: New mode for DO1 (=0:DISABLED, =1:ENABLED) Bit 1: New mode for DO2 (=0:DISABLED, =1:ENABLED) ... Bit 13: New mode for DO14 (=0:DISABLED, =1:ENABLED) Bit 14: New mode for DO15 (=0:DISABLED, =1:ENABLED)				
GET CONFIG DIGITAL OUTPUTS ENABLE SHORT CUT TO VDD DETECTION WHILE OFF	ASCII READ COMMAND	#GCDOESVDDS<CR> Result: #GCDOESDDS:<ShortCutDOSDec>,<ShortCutDOSHex><CR>	ASCII	
	TX	#255,GCDOESVDDS<CR>		
	RX	#255,GCDOESVDDS:0,0x0<CR>		
		Init & watchdog configuration for shortcut to VDD diagnostic while OFF of digital outputs:		
		DO1-DO15:000.0000.0000.0000		
Returns the actual mode for shortcut to VDD diagnostic while digital output is OFF as decimal number and as hexadecimal number. This values are used after power on of the module an after a watchdog event. ShortCutDOSDec, ShortCutDOSHex The current mode for shortcut diagnostic while OFF of the digital outputs: Bit 0: Open wire diagnostic mode of DO1 (=0:DISABLED, =1:ENABLED) Bit 1: Open wire diagnostic mode of DO2 (=0:DISABLED, =1:ENABLED) ... Bit 13: Open wire diagnostic mode of DO14 (=0:DISABLED, =1:ENABLED) Bit 14: Open wire diagnostic mode of DO15 (=0:DISABLED, =1:ENABLED)				

The configuration process for the other diagnostic functionality in case of power on or IO communication watchdog is similar and found in our command & register list for every product.

You can define the same bits with MODBUS using the following registers:

DIGITAL OUTPUTS:ENABLE OPEN WIRE DETECTION WHILE ON						
INTIAL & WATCHDOG SETUP ENABLE OPEN WIRE DETECTION WHILE ON DO1-DO15	3x59002 4x59002 I:59001	????		0x7FFF	UINT16 R/W	NO
		Initial setup of open wire detection while ON for DO1:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO2:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO3:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO4:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO5:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO6:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO7:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO8:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO9:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO10:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO11:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO12:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO13:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO14:0=DISABLED		1		
		Initial setup of open wire detection while ON for DO15:0=DISABLED		1		
Current FRAM setting for intial and watchdog state for open wire detection while ON for digital output DOx. This state is used after power on and after a communcation watchdog timeout, if a watchdog time is confgured Bit 0: =0:Open wire detection for DO1 is DISABLED, =1:Open wire detection for DO1 is ENABLED Bit 1: =0:Open wire detection for DO2 is DISABLED, =1:Open wire detection for DO2 is ENABLED ... Bit 13: =0:Open wire detection for DO14 is DISABLED, =1:Open wire detection for DO14 is ENABLED Bit 14: =0:Open wire detection for DO15 is DISABLED, =1:Open wire detection for DO15 is ENABLED Write on this register sets all digital outputs to a new state for module restart and watchdog function. The state is saved in FRAM						
DIGITAL OUTPUTS:ENABLE OPEN WIRE DETECTION WHILE OFF						
INTIAL & WATCHDOG SETUP ENABLE OPEN WIRE DETECTION WHILE OFF DO1-DO15	3x59003 4x59003 I:59002	????		0x7FFF	UINT16 R/W	NO
		Initial setup of open wire detection while OFF for DO1:0=DISABLED		1		
		Initial setup of open wire detection while OFF for DO2:0=DISABLED		1		
		Initial setup of open wire detection while OFF for DO3:0=DISABLED		1		
		Initial setup of open wire detection while OFF for DO4:0=DISABLED		1		
		Initial setup of open wire detection while OFF for DO5:0=DISABLED		1		
		Initial setup of open wire detection while OFF for DO6:0=DISABLED		1		
		Initial setup of open wire detection while OFF for DO7:0=DISABLED		1		

		Initial setup of open wire detection while OFF for DO8:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO9:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO10:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO11:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO12:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO13:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO14:0=DISABLED	1		
		Initial setup of open wire detection while OFF for DO15:0=DISABLED	1		

Current FRAM setting for intial and watchdog state for open wire detection while OFF for digital output DOx. This state is used after power on and after a communcation watchdog timeout, if a watchdog time is confgured
Bit 0: =0:Open wire detection for DO1 is DISABLED, =1:Open wire detection for DO1 is ENABLED
Bit 1: =0:Open wire detection for DO2 is DISABLED, =1:Open wire detection for DO2 is ENABLED
...
Bit 13: =0:Open wire detection for DO14 is DISABLED, =1:Open wire detection for DO14 is ENABLED
Bit 14: =0:Open wire detection for DO15 is DISABLED, =1:Open wire detection for DO15 is ENABLED

Write on this register sets all digital outputs to a new state for module restart and watchdog function. The state is saved in FRAM

DIGITAL OUTPUTS:ENABLE SHORTCUT DETECTION WHILE OFF

INTIAL & WATCHDOG SETUP ENABLE SHORTCUT TO VDD DETECTION WHILE OFF DO1-DO15	3x59004 4x59004 I:59003	????		0x7FFF	UINT16 R/W	NO
		Initial setup of shortcut to VDD detection while OFF for DO1:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO2:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO3:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO4:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO5:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO6:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO7:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO8:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO9:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO10:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO11:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO12:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO13:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO14:0=DISABLED		1		
		Initial setup of shortcut to VDD detection while OFF for DO15:0=DISABLED		1		

Current FRAM setting for intial and watchdog state for shortcut to VDD detection while OFF for digital output DOx. This state is used after power on and after a communcation watchdog timeout, if a watchdog time is confgured
Bit 0: =0:Shortcut detection for DO1 is DISABLED, =1:Shortcut detection for DO1 is ENABLED
Bit 1: =0:Shortcut detection for DO2 is DISABLED, =1:Shortcut detection for DO2 is ENABLED
...
Bit 13: =0:Shortcut detection for DO14 is DISABLED, =1:Shortcut detection for DO14 is ENABLED
Bit 14: =0:Shortcut detection for DO15 is DISABLED, =1:Shortcut detection for DO15 is ENABLED

Write on this register sets all digital outputs to a new state for module restart and watchdog function. The state is saved in FRAM

8.3.3 Relay outputs ≦30V=, ≦250V~, ≦6A, $AgSnO_2$

This output type supports relay outputs for maximum 30Vdc or 250Vac voltage and max. 6A current. All relay are normally open relay with form A contacts. As contact material we use only relays with $AgSnO_2$.

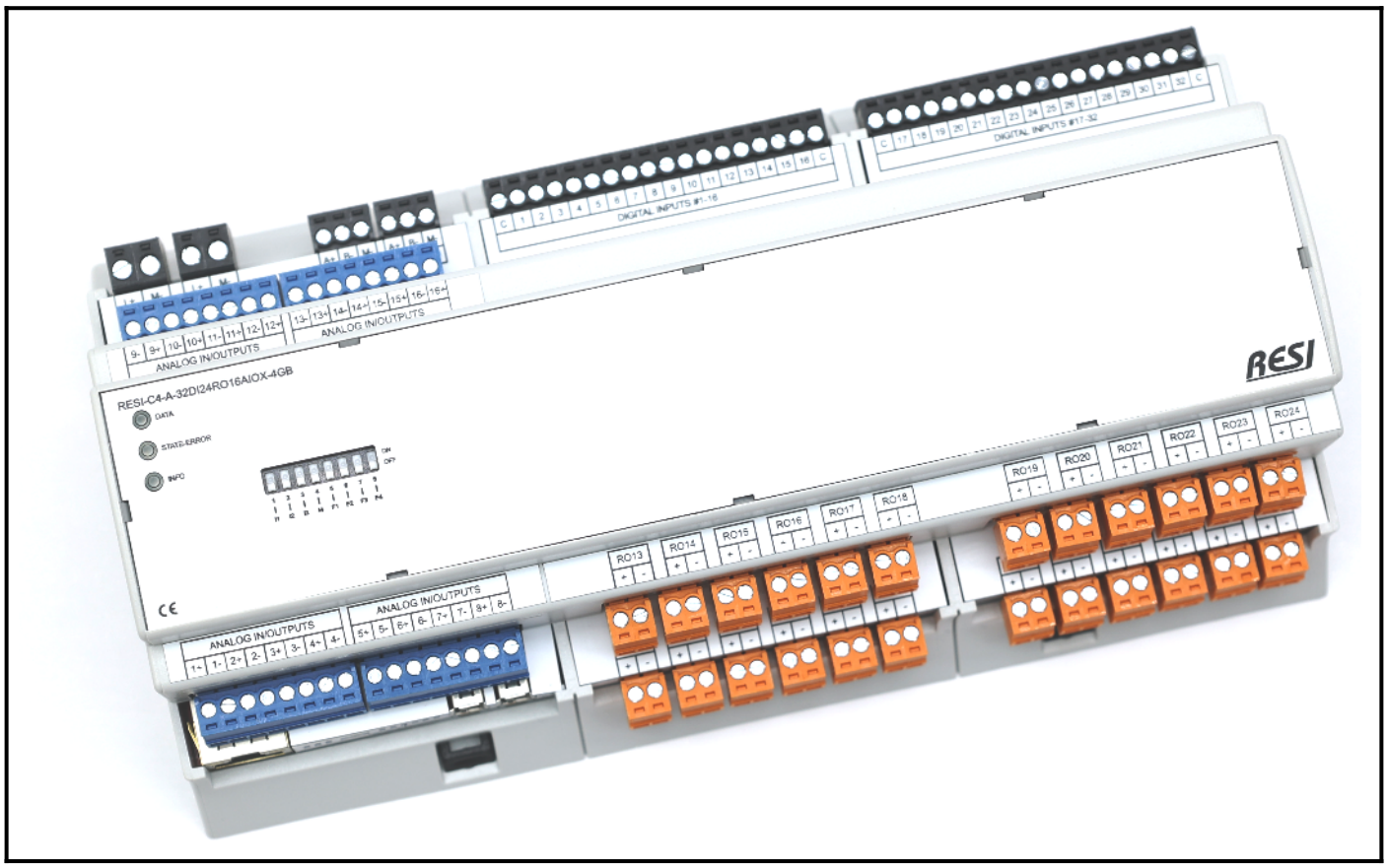
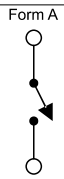


Figure: Our RESI-C4-A-32DI24RO16AIOX IoT controller with 24 relay outputs

8.3.3.1 Technical specification

The realy outputs meets the following technical specification

RELAY OUTPUTS	
Update rate	As fast as possible
Relay type	Mono stable, Form A
	
Maximum voltage	250Vac
Maximum current	6A
Mechanical lifetime	10^6 cycles of operation
Contact material	$AgSnO_2$
Max. switching power AC1	1500VA
Max. switching power AC15 (230V~)	300VA
Max. switching power AC3	185W
Max. switching power DC1	6A@30V=
	0.2A@110V=
	0.12A@220V=
Insulation	Creep-age and clearance distance 8mm
Cable connection	Power supply: via one 2-pin plug in terminal block
Terminal type	RM3.5
Galvanic insulation	Yes, with the relay

8.3.3.2 Additional terminals or functionalities

Depending on the module the relay outputs are grouped in 12 or 24 relay outputs on one terminal block

RELAY OUTPUTS		
Relay output x	Terminal type:	RM3.5
	ROx:+:	Contact A of Form A relay
	ROx:-:	Contact B of Form A relay
	1..n:	Relay output 1-n
		0=Relay is open
		1=Relay is closed
Pin layout	12/24 Form A relay outputs	
	Twelve/twenty-four 2 pin plug-in terminal block	
	Pin 1:	1: Relay output #1: Contact A
	Pin 2:	2: Relay output #1: Contact B
	Pin 1:	1: Relay output #12/24: Contact A
	Pin 2:	2: Relay output #12/24: Contact B
INFO	If at least one of the relay outputs is activated (ON), this LED is ON.	
	If none of the relay outputs are activated (OFF), this LED is OFF.	

8.3.3.3 Cabling of the relay outputs

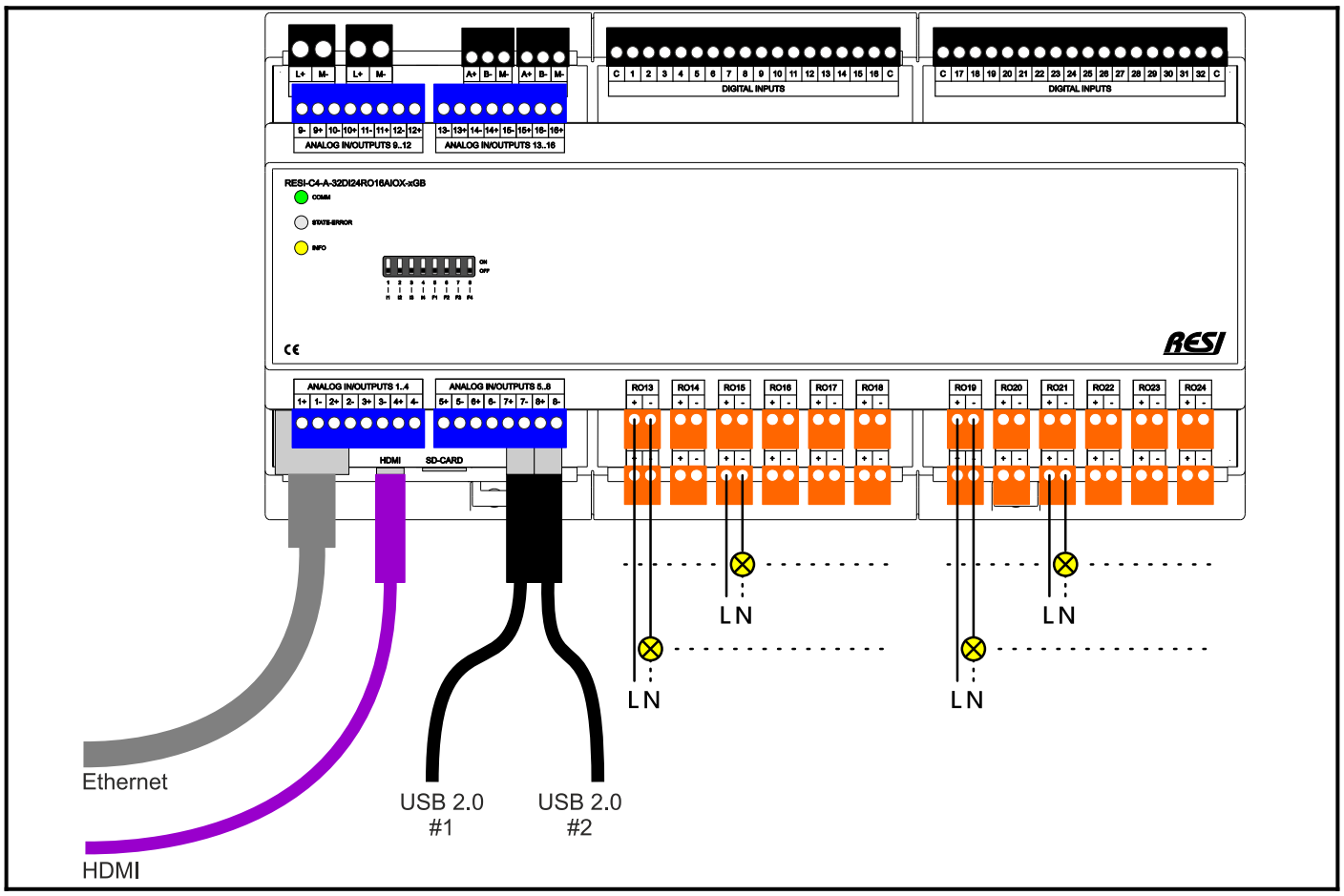


Figure: Example of cabling of the relay outputs to a RESI-C4-A-32DI24RO16AIOX IoT controller

8.3.3.4 Using the relay outputs with ASCII+MODBUS

8.3.3.4.1 Update all digital inputs & relay outputs

In ASCII there is a special command to update all relay outputs and read back the actual state of all inputs with one command:

UPDATE DIGITAL INPUTS AND OUTPUTS	ASCII WRITE COMMAND	#UDIOS:<OutAllDOS><CR> Result: #UDIOS:<InAllDISDec>,<InAllDISHex><CR>	ASCII	YES
	DO1	0:OFF		
	DO2	0:OFF		
	DO3	0:OFF		
	DO4	0:OFF		
	DO5	0:OFF		
	DO6	0:OFF		
	DO7	0:OFF		
	DO8	0:OFF		
	DO9	0:OFF		
	DO10	0:OFF		
	DO11	0:OFF		
	DO12	0:OFF		
	TX	#255,UDIOS:0<CR>		
	RX	#255,UDIOS:2147483648,0x80000000<CR>		
		Actual status of digital inputs:1000.0000.0000.0000.0000.0000.0000.0000		

Sets all digital outputs to the new state OutAllDOS and gives back the current status of all digital inputs InAllDIS as decimal and hexadecimal value
OutAllDOS: The new state for all digital outputs
Bit 0: State of DO1 (=0:OFF, =1:ON)
Bit 1: State of DO2 (=0:OFF, =1:ON)
Bit 2: State of DO3 (=0:OFF, =1:ON)
...
Bit 9: State of DO10 (=0:OFF, =1:ON)
Bit 10: State of DO11 (=0:OFF, =1:ON)
Bit 11: State of DO12 (=0:OFF, =1:ON)

InAllDIS: The current state for all digital inputs
Bit 0: State of DI1 (=0:OFF, =1:ON)
Bit 1: State of DI2 (=0:OFF, =1:ON)
Bit 2: State of DI3 (=0:OFF, =1:ON)
...
Bit 29: State of DI30 (=0:OFF, =1:ON)
Bit 30: State of DI31 (=0:OFF, =1:ON)
Bit 31: State of DI32 (=0:OFF, =1:ON)

8.3.3.4.2 Current status of relay outputs

In ASCII you can read the current status of the digital outputs with the commands GDOS or GDOx:

GET DIGITAL OUTPUTS	ASCII READ COMMAND	#GDOS<CR> Result: #GDOS:<DOSDec>,<DOSHex><CR>	ASCII	
	TX	#255,GDOS<CR>		
	RX	#255,GDOS:0,0x0<CR>		
		Actual status of digital outputs:0000.0000.0000		

Returns the actual state of the digital outputs as decimal number and as hexadecimal number.
DOSDec, DOSHex
The current state of the digital outputs:
Bit 0: State of DO1 (=0:OFF, =1:ON)
Bit 1: State of DO2 (=0:OFF, =1:ON)
Bit 2: State of DO3 (=0:OFF, =1:ON)
...
Bit 9: State of DO10 (=0:OFF, =1:ON)
Bit 10: State of DO11 (=0:OFF, =1:ON)
Bit 11: State of DO12 (=0:OFF, =1:ON)

GET DIGITAL OUTPUT DOx	ASCII READ COMMAND	#GDO<DONR><CR> Result: #GDO<DONR>:<DOxDec>,<DOxHex><CR>	ASCII	
	DONR	2		
	TX	#255,GDO2<CR>		
	RX	#255,GDO2:0,0x0<CR>		
		Actual status of digital output DO2:0=OFF		

Returns the actual state of the digital output DOx as decimal number and as hexadecimal number.
DOxDec, DOxHex
The current state of the digital output DOx:
=0: relay output is OFF
=1: relay output is ON

For writing to a digital output in ASCII use the command SDOS or SDOx:

SET DIGITAL OUTPUTS	ASCII WRITE COMMAND	#SDOS:<OutAllDOS><CR> Result: #OK<CR>	ASCII	YES
	DO1	0:OFF		
	DO2	0:OFF		
	DO3	0:OFF		
	DO4	0:OFF		
	DO5	0:OFF		
	DO6	0:OFF		
	DO7	0:OFF		
	DO8	0:OFF		
	DO9	0:OFF		
	DO10	0:OFF		
	DO11	0:OFF		
	DO12	0:OFF		
	TX	#255,SDOS:0<CR>		
	RX	#255,OK<CR>		

Sets all digital outputs to the new state OutAllDOS
The new state for all digital outputs
Bit 0: State of DO1 (=0:OFF, =1:ON)
Bit 1: State of DO2 (=0:OFF, =1:ON)
Bit 2: State of DO3 (=0:OFF, =1:ON)
...
Bit 9: State of DO10 (=0:OFF, =1:ON)
Bit 10: State of DO11 (=0:OFF, =1:ON)
Bit 11: State of DO12 (=0:OFF, =1:ON)

SET DIGITAL OUTPUT DOx	ASCII WRITE COMMAND	#SDO<DONR>:<Out><CR> Result: #OK<CR>	ASCII	NO
	DONR	2		
	DOx	0:OFF		
	TX	#255,SDO2:0<CR>		
	RX	N/A		

<DONR>: 1=DO1..12=DO12

Sets the new state for digital output DOx. The state is defined with <Out>.
Out
The new state of the digital output DOx:
=0: digital output is OFF
=1: digital output is ON

In MODBUS you have many coils and registers which will show the actual digital output state or with which you can set a new output state:

Here are registers for coils or registers (Every output as one bit):

STATUS DIGITAL OUTPUTS						
DO1	1x00017 2x00017 I:16	????		1	BIT R/W	NO
		Actual state of DO1:0=OFF		ENTER NEW STATE (0 or 1)		
Current state of the digital output DOx =0:DO is OFF, =1:DO is ON Writing on this register changes the state of the digital output						
DO2	1x00018 2x00018 I:17	????		0	BIT R/W	NO
		Actual state of DO2:0=OFF		ENTER NEW STATE (0 or 1)		

The same reading and writing can be done by holding registers:

STATUS DIGITAL OUTPUTS						
DO1	3x00017 4x00017 I:16	0,0x0000 B:00 00		1	UINT16 R/W	NO
		Actual state of DO1:0=OFF		ENTER NEW STATE (0 or 1)		
Current state of the digital output DOx =0:DO is OFF, =1:DO is ON Writing on this register changes the state of the digital output						
DO2	3x00018 4x00018 I:17	0,0x0000 B:00 00		0	UINT16 R/W	NO
		Actual state of DO2:0=OFF		ENTER NEW STATE (0 or 1)		

But you can also read and write to all digital outputs together:

STATUS OF DIGITAL OUTPUTS						
STATUS OF ALL DOS DO1-DO12	3x10004 4x10004 I:10003	0,0x0000 B:00 00		0x0FFF	UINT16 R/W	NO
		Actual state of DO1:0=OFF		1		
		Actual state of DO2:0=OFF		1		
		Actual state of DO3:0=OFF		1		
		Actual state of DO4:0=OFF		1		
		Actual state of DO5:0=OFF		1		
		Actual state of DO6:0=OFF		1		
		Actual state of DO7:0=OFF		1		
		Actual state of DO8:0=OFF		1		
		Actual state of DO9:0=OFF		1		
		Actual state of DO10:0=OFF		1		
		Actual state of DO11:0=OFF		1		
		Actual state of DO12:0=OFF		1		
Actual state of all digital outputs Bit 0: =0:DO1 is OFF, =1:DO1 is ON Bit 1: =0:DO2 is OFF, =1:DO2 is ON ... Bit 10: =0:DO11 is OFF, =1:DO11 is ON Bit 11: =0:DO12 is OFF, =1:DO12 is ON Write on this register sets all digital outputs to a new state						

8.3.3.4.3 Pulsing the relay outputs

A special ASCII command generates a one time pulse on a digital output:

DIGITAL OUTPUTS: PULSE OUTPUT			
PULSE DOx	ASCII WRITE COMMAND	#PDO<DONR>:<Time><CR> Result: #OK<CR>	ASCII	YES
	DONR	2		
	TIME	200		
	TX	#255,PDO2:200<CR>		
	RX	#255,OK<CR>		
<DONR>: 1=DO1..12=DO12				
<Time>: 0..65535*100ms				
This command switches the digital output DOx on for the pulse duration <PulseTimeIn100ms>*100ms. PulseTimeIn100ms: A duration in 100ms units. The corresponding digital output is switched on for this time period.				
GET PULSE TIMER DOx	ASCII READ COMMAND	#GPT<DONR><CR> Result: #GPT:<TimeDec>,<TimeHex><CR>	ASCII	
	DONR	2		
	TX	#255,GPT2<CR>		
	RX	#255,GPT2:19812,0x4D64<CR>		
		Actual pulse time for DO2:19,8s		
<DONR>: 1=DO1..12=DO12				
Returns the remaining timer value of the pulse for digital output DOx in ms. PulseTimeInMSDec, PulseTimeInMSHex The remaining time of the pulse in Milliseconds				

But you can also initiate this digital output pulse with this MODBUS registers:

PULSE TIME FOR DIGITAL OUTPUTS						
PULSE TIME DO1	3x20001 4x20001 I:20000	0,0x0000 B:00 00	200	20,0	UINT16 R/W	YES
Generate a pulse on digital output x in 100ms units (0,1 to 6553,5 Seconds selectable) If you write onto this register, the digital output will be switched on for the desired time in 100ms units.						
PULSE TIME DO2	3x20002 4x20002 I:20001	0,0x0000 B:00 00	300	30,0	UINT16 R/W	NO
PULSE TIME DO3	3x20003 4x20003 I:20002	0,0x0000 B:00 00	400	40,0	UINT16 R/W	NO
PULSE TIME DO4	3x20004 4x20004 I:20003	0,0x0000 B:00 00	500	50,0	UINT16 R/W	NO

The remaining time for the current pulse can be read with this registers:

PULSE STATUS FOR DIGITAL OUTPUTS						
PULSE TIMER DO1	3x21001 4x21001 I:21000	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		0,0 seconds				
Remaining time of the pulse on digital output x in Milliseconds.						
PULSE TIMER DO2	3x21003 4x21003 I:21002	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		0,0 seconds				
PULSE TIMER DO3	3x21005 4x21005 I:21004	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		0,0 seconds				

PULSE STATUS FOR DIGITAL OUTPUTS						
PULSE TIMER DO1	3x21031 4x21031 I:21030	0,0x00000000 B:00 00 00 00			UINT32R R/O	
		0,0 seconds				
Remaining time of the pulse on digital output x in Milliseconds.						
PULSE TIMER DO2	3x21033 4x21033 I:21032	0,0x00000000 B:00 00 00 00			UINT32R R/O	
		0,0 seconds				

8.3.4 Universal analog inputs & outputs 0-10V, 0-20mA, RTD

Our IoT controller with universal analog inputs and output offer versatile use of analog signals in the field. Every analog input or output can be configured with software to its function. You can choose between the following analog IO types:

- UNUSED: Analog IO is not used in your application

- ANALOG INPUT 0-10V or 2-10V
- ANALOG INPUT 0-20mA or 4-20mA loop powered
- ANALOG INPUT 0-20mA or 4-20mA external powered

- ANALOG OUTPUT 0-10V or 2-10V
- ANALOG OUTPUT 0-20mA or 4-20mA

- RESISTOR INPUT ohm measurement 0-1MOhm
- TEMPERATURE INPUT for PT100 sensor
- TEMPERATURE INPUT for PT1000 sensor
- TEMPERATURE INPUT for NI1000-DIN43760 sensor

- DIGITAL INPUT for logic input 24Vdc
- DIGITAL INPUT loop powered

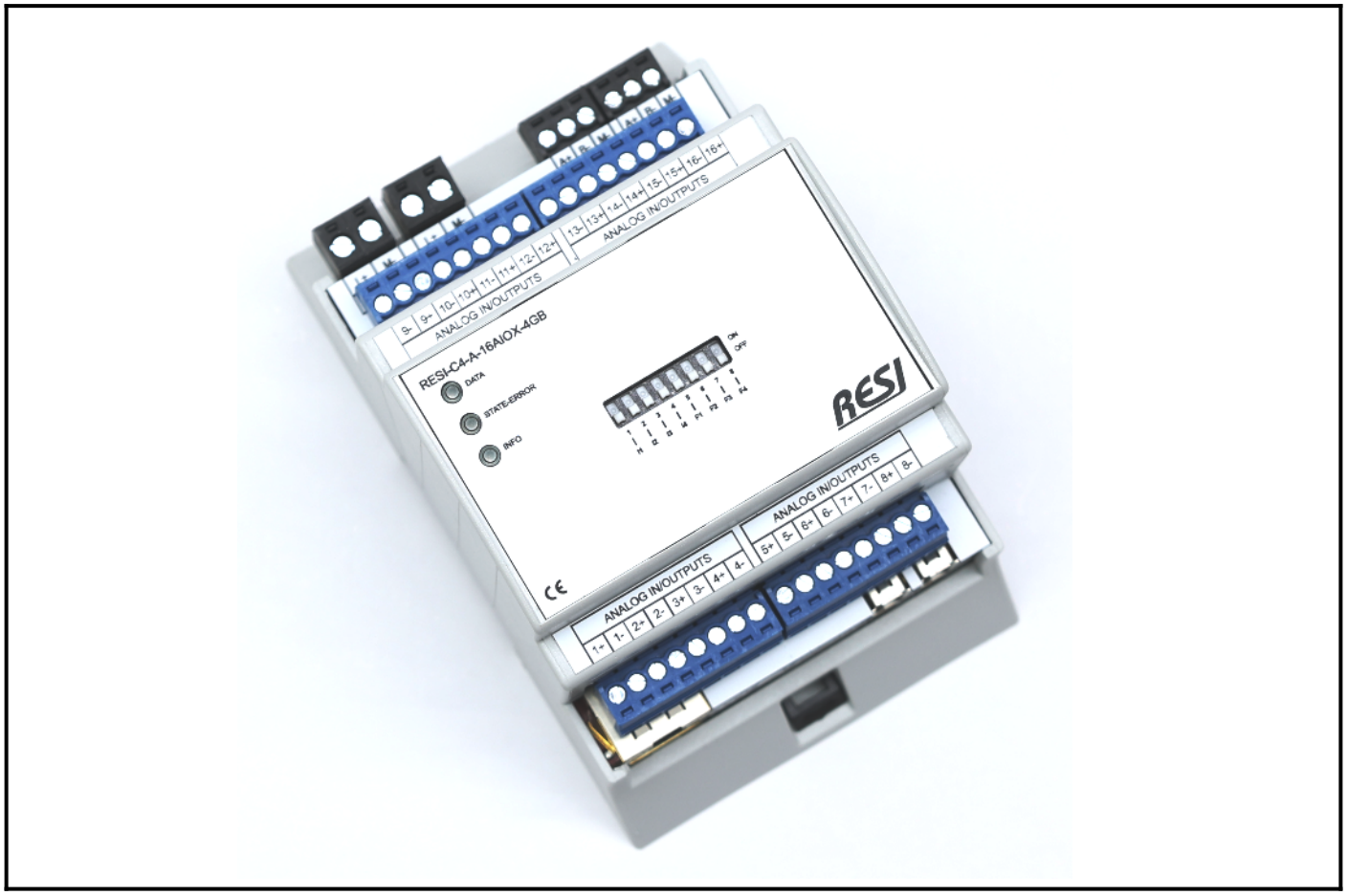


Figure: Our RESI-C4-A-16AIOX IoT controller with 16 universal analog inputs & outputs

8.3.4.1 Technical specification

The analog inputs and outputs meet the following technical specification

ANALOG INPUTS/OUTPUTS		
VOLTAGE INPUT		
Input range	0-10V	
Resolution	16 bit	
Accuracy	max. ±0.04V	
CURRENT INPUT		
Input range	0-25mA	
Resolution	16 bit	
Accuracy	max. ±0.125mA	
VOLTAGE OUTPUT		
Output range	0-11V	
Resolution	13 bit	
Accuracy	max. ±0.044V	
CURRENT OUTPUT		
Output range	0-25mA	
Resolution	13 bit	
Accuracy	max. ±0.1375mA	
RESISTOR (RTD) INPUT		
Input range	0-1MOhm	
Resolution	16 bit	
Accuracy	Range	Accuracy
	0-80Ω	±0.5%±0.5Ω
	80-200Ω	±0.3%
	200-1kΩ	±0.2%
	1k-10kΩ	±0.2%
	10k-20kΩ	±0.3%
	20k-100kΩ	±0.8%
	100k-200kΩ	±1.0%
	200k-1MΩ	±8%

TEMPERATURE INPUT

for 2 wire PT100, PT1000, NI1000-DIN43760 sensors with internal linearisation

	Sensor	Accuracy
Sensor type	PT100	±0.3%
	PT1000	±0.2%
	NI1000-DIN43760	±0.2%

DIGITAL INPUT	Logic input
Input range	≦40V, 1,8mA
Input Threshold	12V= hysteresis 11.80-12.00V

DIGITAL INPUT	Loop powered
Input current if contact is closed	4mA

Terminal type	RM3.5
Galvanic insulation	All grounds of all analog inputs and outputs are internally tied together.
from the rest of the IoT controller	The complete analog input & output block is galvanically isolated

8.3.4.2 Additional terminals or functionalities

UNIVERSAL ANALOG INPUTS & OUTPUTS		
IO groups	Terminal type:	RM3.5
	N+:	Signal + for analog input or output N
	N-:	Signal – or ground for analog input or output N
Pin layout	4 universal analog inputs or outputs	
	One to four 8 pin plug-in terminal blocks	
	Pin 1:	1+,5+,9+,13+: Signal + for AI#1,5,9 or 13
	Pin 2:	1-,5-,9-,13-: Signal – or ground for AI#1,5,9 or 13
	…	
	Pin 1:	4+,8+,12+,16+: Signal + for AI#4,8,12 or 16
	Pin 2:	4-,8-1,12-,16-: Signal – or ground for AI#4,8,12 or 16

8.3.4.3 Cabling of the universal analog inputs or outputs

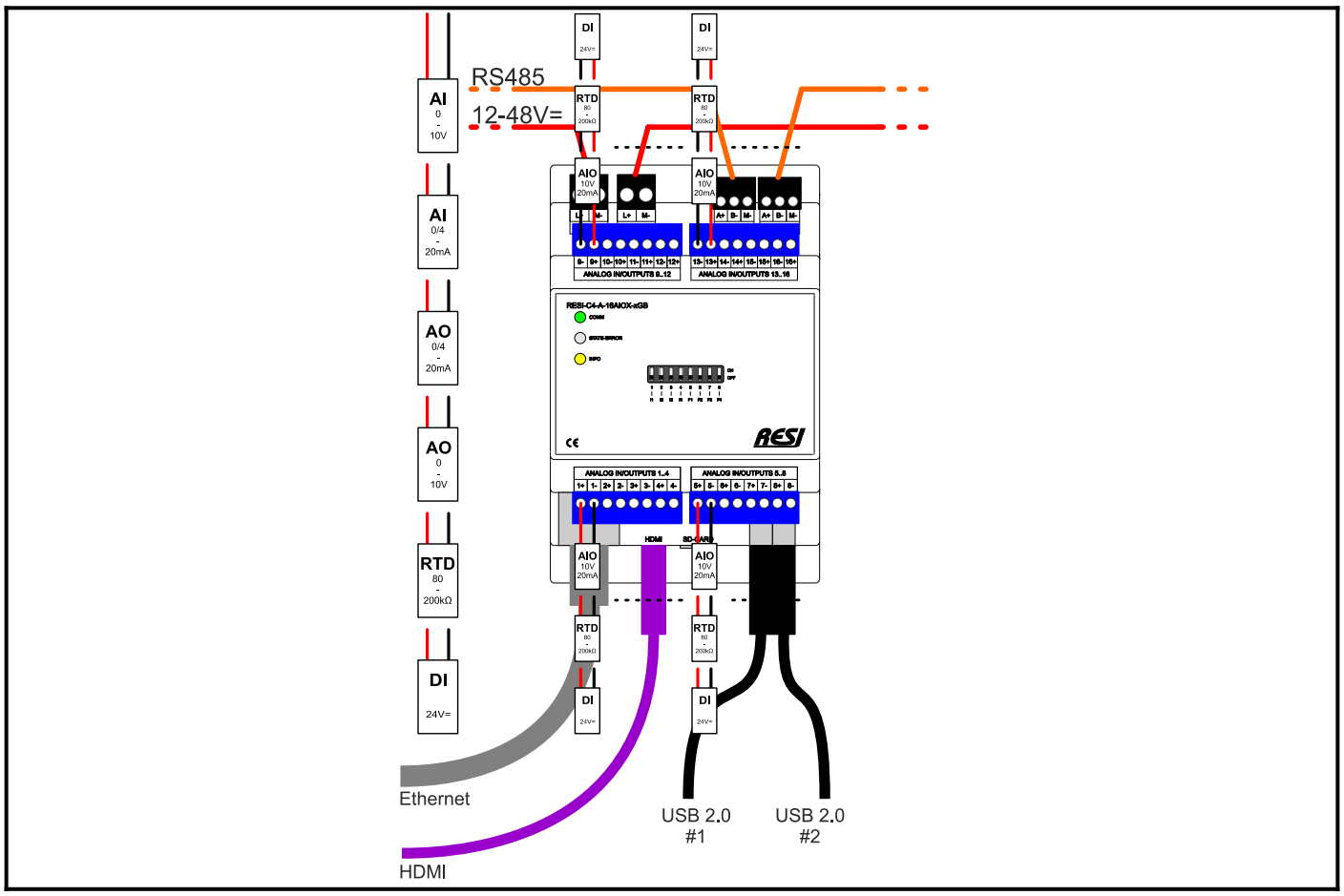


Figure: Example of cabling of the universal analog inputs or outputs to a RESI-C4-A-16AIOX IoT controller

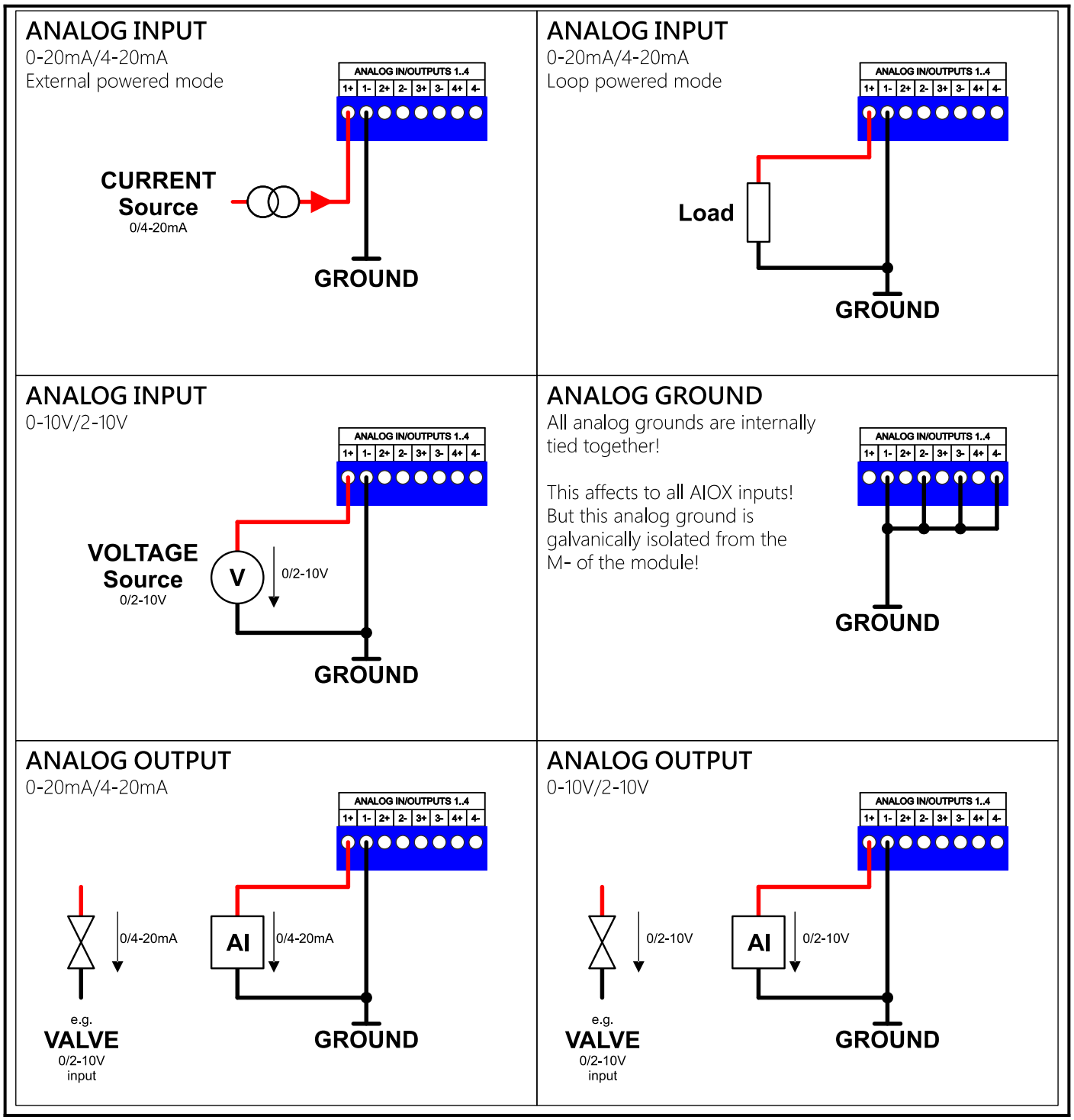


Figure: Wiring of the different analog input and output signals and types

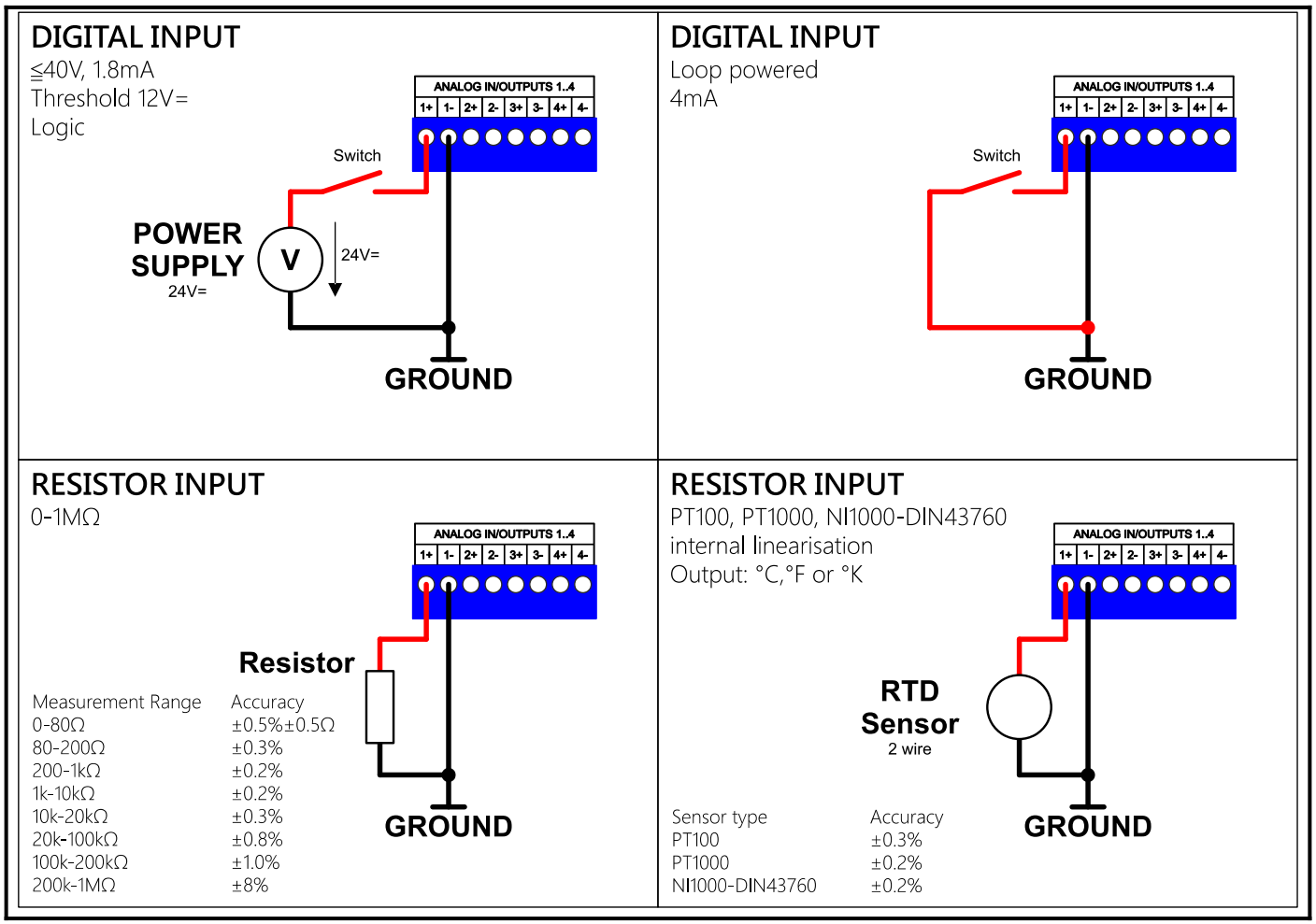


Figure: Wiring of the different digital and temperature input types

8.3.4.4 Using the universal analog inputs & outputs with ASCII+MODBUS

8.3.4.4.1 Communication with co-processor

The universal analog inputs and outputs (AIOX) are driven be an additional ARM processor. This processor talks to the ARM co-processor via an internal serial interface. The two processors exchange the current status of all analog inputs and outputs every 100ms. So the effective update rate on the AIOX is 10 samples per second.

8.3.4.4.2 Howto set the IO type of the AIOX

In ASCII you can set the IO type for every AIOX with the commands SIOTYPS or SIOTYPx. This command will also save the actual IO type for every AIOX in the ferromagnetic RAM. After a restart of the IoT controller the last IO type setting is automatically used for all AIOX inputs and outputs.

AIOIX CONFIGURATION				
SET IO TYPES	ASCII WRITE COMMAND	#SIOTYPS:<IOTyp1>,<IOTyp2>,<IOTyp3>,<IOTyp4>,<IOTyp5>,<IOTyp6>,<IOTyp7>,<IOTyp8>,<IOTyp9>,<IOTyp10>,<IOTyp11>,<IOTyp12>,<IOTyp13>,<IOTyp14>,<IOTyp15>,<IOTyp16><CR> Result: #OK<CR>	ASCII	YES
	IOTyp1	VO[0-10V]		
	IOTyp2	VO[0-10V]		
	IOTyp3	VO[0-10V]		
	IOTyp4	VO[0-10V]		
	IOTyp5	VO[0-10V]		
	IOTyp6	VO[0-10V]		
	IOTyp7	VO[0-10V]		
	IOTyp8	VO[0-10V]		
	IOTyp9	VO[0-10V]		
	IOTyp10	VO[0-10V]		
	IOTyp11	VO[0-10V]		
	IOTyp12	VO[0-10V]		
	IOTyp13	VO[0-10V]		
	IOTyp14	VO[0-10V]		
	IOTyp15	VO[0-10V]		
	IOTyp16	VO[0-10V]		
	TX	#255,SIOTYPS:VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V]<CR>		
	RX	N/A		
This command defines for all 16 universal IOs a new type of IO: IOTypx stands for the new type: UU: Unused – high impedance VI[0-10V]: VOLTAGE INPUT for 0 to 10V Signals VI[2-10V]: VOLTAGE INPUT for 2 to 10V Signals VO[0-10V]: VOLTAGE OUTPUT for 0 to 10V Signals VO[2-10V]: VOLTAGE OUTPUT for 2 to 10V Signals] CI[0-20mA;LP]: CURRENT INPUT for 0 to 20mA Signals – loop powered CI[4-20mA;LP]: CURRENT INPUT for 4 to 20mA Signals – loop powered CI[0-20mA;EP]: CURRENT INPUT for 0 to 20mA Signals – external powered CI[4-20mA;EP]: CURRENT INPUT for 4 to 20mA Signals – external powered CO[0-20mA]: CURRENT OUTPUT for 0 to 20mA Signals CO[4-20mA]: CURRENT OUTPUT for 4 to 20mA Signals RTDI[OHM]: RTD SENSOR INPUTfor Ohm measurement between 0 and 1MOhm DI[24V;L]: DIGITAL INPUT for 24Vdc – logic, threshold 12V DI[24V;LP]: DIGITAL INPUT for 24Vdc – loop powered HINT: The last IO type is automatically stored in FRAM and will be used after a system restart.				
SET IO TYPx	ASCII WRITE COMMAND	#SIOTYP<IONR>:<IOTypx><CR> Result: #OK<CR>	ASCII	NO
	IONR	1		
	IOTypx	VI[0-10V]		
	TX	#255,SIOTYP1:VI[0-10V]<CR>		
	RX	N/A		
This command defines for the universal IO IONR a new type of IO: IOTypx stands for the new type: UU: Unused – high impedance VI[0-10V]: VOLTAGE INPUT for 0 to 10V Signals VI[2-10V]: VOLTAGE INPUT for 2 to 10V Signals VO[0-10V]: VOLTAGE OUTPUT for 0 to 10V Signals VO[2-10V]: VOLTAGE OUTPUT for 2 to 10V Signals] CI[0-20mA;LP]: CURRENT INPUT for 0 to 20mA Signals – loop powered CI[4-20mA;LP]: CURRENT INPUT for 4 to 20mA Signals – loop powered CI[0-20mA;EP]: CURRENT INPUT for 0 to 20mA Signals – external powered CI[4-20mA;EP]: CURRENT INPUT for 4 to 20mA Signals – external powered CO[0-20mA]: CURRENT OUTPUT for 0 to 20mA Signals CO[4-20mA]: CURRENT OUTPUT for 4 to 20mA Signals RTDI[OHM]: RTD SENSOR INPUTfor Ohm measurement between 0 and 1MOhm DI[24V;L]: DIGITAL INPUT for 24Vdc – logic, threshold 12V DI[24V;LP]: DIGITAL INPUT for 24Vdc – loop powered HINT: The last IO type is automatically stored in FRAM and will be used after a system restart.				

You can read back the current setting for all AIOX with GIOTYPS or for one channel with GIOTYPx:

GET IO TYPES	ASCII READ COMMAND	#GIOTYPS<CR> Result: #GIOTYPS:<IOTyp1Txt>,<IOTyp2Txt>,...,<IOTyp16Txt><CR>	ASCII	
	TX	#255,GIOTYPS<CR>		
	RX	#1,GIOTYPS:VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V],VO[0-10V]<CR>		
		Actual type of IO1:VO[0-10V]		
		Actual type of IO2:VO[0-10V]		
		Actual type of IO3:VO[0-10V]		
		Actual type of IO4:VO[0-10V]		
		Actual type of IO5:VO[0-10V]		
		Actual type of IO6:VO[0-10V]		
		Actual type of IO7:VO[0-10V]		
		Actual type of IO8:VO[0-10V]		
		Actual type of IO9:VO[0-10V]		
		Actual type of IO10:VO[0-10V]		
		Actual type of IO11:VO[0-10V]		
		Actual type of IO12:VO[0-10V]		
		Actual type of IO13:VO[0-10V]		
		Actual type of IO14:VO[0-10V]		
		Actual type of IO15:VO[0-10V]		
		Actual type of IO16:VO[0-10V]		

This command shows for all 16 universal IOs the current selected type of IO:
IOTypx stands for the types:
UU: Unused – high impedance
VI[0-10V]: VOLTAGE INPUT for 0 to 10V Signals
VI[2-10V]: VOLTAGE INPUT for 2 to 10V Signals
VO[0-10V]: VOLTAGE OUTPUT for 0 to 10V Signals
VO[2-10V]: VOLTAGE OUTPUT for 2 to 10V Signals]
CI[0-20mA;LP]: CURRENT INPUT for 0 to 20mA Signals – loop powered
CI[4-20mA;LP]: CURRENT INPUT for 4 to 20mA Signals – loop powered
CI[0-20mA;EP]: CURRENT INPUT for 0 to 20mA Signals – external powered
CI[4-20mA;EP]: CURRENT INPUT for 4 to 20mA Signals – external powered
CO[0-20mA]: CURRENT OUTPUT for 0 to 20mA Signals
CO[4-20mA]: CURRENT OUTPUT for 4 to 20mA Signals
RTDI[OHM]: RTD SENSOR INPUTfor Ohm measurement between 0 and 1MOhm
DI[24V;L]: DIGITAL INPUT for 24Vdc – logic, threshold 12V
DI[24V;LP]: DIGITAL INPUT for 24Vdc – loop powered

GET IO TYPx	ASCII READ COMMAND	#GIOTYP<IONR><CR> Result: #GIOTYP<IONR>:<IOTypxTxt><CR>	ASCII	
	IONR	1		
	TX	#255,GIOTYP1<CR>		
	RX	#1,GIOTYP1:VO[0-10V]<CR>		
		Actual type of IO1:VO[0-10V]		

This command shows for the universal IO IONR the current selected type:
IOTypx stands for the types:
UU: Unused – high impedance
VI[0-10V]: VOLTAGE INPUT for 0 to 10V Signals
VI[2-10V]: VOLTAGE INPUT for 2 to 10V Signals
VO[0-10V]: VOLTAGE OUTPUT for 0 to 10V Signals
VO[2-10V]: VOLTAGE OUTPUT for 2 to 10V Signals]
CI[0-20mA;LP]: CURRENT INPUT for 0 to 20mA Signals – loop powered
CI[4-20mA;LP]: CURRENT INPUT for 4 to 20mA Signals – loop powered
CI[0-20mA;EP]: CURRENT INPUT for 0 to 20mA Signals – external powered
CI[4-20mA;EP]: CURRENT INPUT for 4 to 20mA Signals – external powered
CO[0-20mA]: CURRENT OUTPUT for 0 to 20mA Signals
CO[4-20mA]: CURRENT OUTPUT for 4 to 20mA Signals
RTDI[OHM]: RTD SENSOR INPUTfor Ohm measurement between 0 and 1MOhm
DI[24V;L]: DIGITAL INPUT for 24Vdc – logic, threshold 12V
DI[24V;LP]: DIGITAL INPUT for 24Vdc – loop powered

In MODBUS you have the following holding registers to configure the type of the analog inputs or outputs:

AIOX IO TYPES						
IO TYPE1	3x40001 4x40001 I:40000	13,0x000D B:00 0D		13:RTDI[OHM]	UINT16 R/W	YES
		Actual IO type of AIOx:13:RTDI[OHM]		CHOOSE NEW IO TYPE FROM LIST		

Current configured IO type for AIOXx
=0: UNUSED
=1: VOLTAGE INPUT[0-10V]
=2: VOLTAGE INPUT[2-10V]
=3: VOLTAGE OUTPUT[0-10V]
=4: VOLTAGE OUTPUT[2-10V]
=5: CURRENT INPUT LOOP POWERED[0-20mA]
=6: CURRENT INPUT LOOP POWERED[4-20mA]
=7: CURRENT INPUT EXTERNAL POWERED[0-20mA]
=8: CURRENT INPUT EXTERNAL POWERED[4-20mA]
=9: CURRENT OUTPUT[0-20mA]
=10: CURRENT OUTPUT[4-20mA]
=11: DIGITAL INPUT LOGIC 24V=
=12: DIGITAL INPUT LOOP POWERED
=13: RESISTANCE MEASUREMENT

HINT: The last IO type is automatically stored in FRAM and will be used after a system restart.

IO TYPE2	3x40002 4x40002 I:40001	12,0x000C B:00 0C		12:DI[24V;LP]	UINT16 R/W	YES
		Actual IO type of AIOx:12:DI[24V;LP]		CHOOSE NEW IO TYPE FROM LIST		

8.3.4.4.3 Howto read analog inputs 0-10V or 2-10V

If an AIOX is configured either to ANALOG INPUT with 0-10V or to ANALOG INPUT with 2-10V you can use the ASCII commands GVISV or GVIVx to read the actual value of the analog input:

VOLTAGE INPUTS				
GET VOLTAGE INPUTS IN VOLT	ASCII READ COMMAND	#GVISV<CR> Result: #GVISV:<IOVolt1Dbl>,<IOVolt2Dbl>,...,<IOVolt16Dbl><CR>	ASCII	
	TX	#255,GVISV<CR>		
	RX	#1,GVISV:999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Actual voltage on IO1:999.99V		
		Actual voltage on IO2:999.99V		
		Actual voltage on IO3:999.99V		
		Actual voltage on IO4:999.99V		
		Actual voltage on IO5:999.99V		
		Actual voltage on IO6:999.99V		
		Actual voltage on IO7:999.99V		
		Actual voltage on IO8:999.99V		
		Actual voltage on IO9:999.99V		
		Actual voltage on IO10:999.99V		
		Actual voltage on IO11:999.99V		
		Actual voltage on IO12:999.99V		
		Actual voltage on IO13:999.99V		
		Actual voltage on IO14:999.99V		
		Actual voltage on IO15:999.99V		
		Actual voltage on IO16:999.99V		
This command shows for all VOLTAGE INPUT IOs the current measurement in Volt. The measurement range is 0.0 to 10.00V. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET VOLTAGE INPUT IN VOLT	ASCII READ COMMAND	#GVIV<IONR><CR> Result: #GVIV<IONR>:<IOxVoltDbl><CR>	ASCII	
	IONR	1		
	TX	#255,GVIV1<CR>		
	RX	#1,GVIV1:999.99<CR>		
		Actual voltage on IO1:999.99V		
This command shows for the VOLTAGE INPUT IO <IONR> the current measurement in Volt. The measurement range is 0.0 to 10.00V. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

For all AIOX, which have a different IO type, this function returns 999.99. Otherwise the last measured analog input will be returned.

But you can read the analog input also as percentage value with the ASCII command GVISP or GVIPx:

GET VOLTAGE INPUTS IN PERCENT	ASCII READ COMMAND	#GVISP<CR> Result: #GVISP:<IOPercent1Dbl>,<IOPercent2Dbl>,...,<IOPercent16Dbl><CR>	ASCII	
	TX	#255,GVISP<CR>		
	RX	#1,GVISP:999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Actual percentage on IO1:999.99%		
		Actual percentage on IO2:999.99%		
		Actual percentage on IO3:999.99%		
		Actual percentage on IO4:999.99%		
		Actual percentage on IO5:999.99%		
		Actual percentage on IO6:999.99%		
		Actual percentage on IO7:999.99%		
		Actual percentage on IO8:999.99%		
		Actual percentage on IO9:999.99%		
		Actual percentage on IO10:999.99%		
		Actual percentage on IO11:999.99%		
		Actual percentage on IO12:999.99%		
		Actual percentage on IO13:999.99%		
		Actual percentage on IO14:999.99%		
		Actual percentage on IO15:999.99%		
		Actual percentage on IO16:999.99%		
This command shows for all VOLTAGE INPUT IOs the current measurement in Percent. The measurement range is 0.0V -> 0.0% to 10.00V -> 100.0%. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET VOLTAGE INPUT IN PERCENT	ASCII READ COMMAND	#GVIP<IONR><CR> Result: #GVIP<IONR>:<IOxPercentDbl><CR>	ASCII	
	IONR	1		
	TX	#255,GVIP1<CR>		
	RX	#1,GVIP1:999.99<CR>		
		Actual percentage on IO1:999.99%		
This command shows for VOLTAGE INPUT IO <IONR> the current measurement in Percent. The measurement range is 0.0V -> 0.0% to 10.00V -> 100.0%. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

On the MODBUS interface you can read the actual value of the analog inputs with this holding registers:

AIOX:VOLTAGE INPUTS						
VOLTAGE INPUT1 IN VOLTS	3x40017 4x40017 I:40016	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of VIx:65535=N/V				
Current value of voltage input in x*100V, range 0-10V =65535,0xFFFF: The channel is not configured as voltage input						
VOLTAGE INPUT2 IN VOLTS	3x40018 4x40018 I:40017	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of VIx:65535=N/V				
VOLTAGE INPUT3 IN VOLTS	3x40019 4x40019 I:40018	0,0x0000 B:00 00			UINT16 R/O	
		Actual value of VIx:0=0,00V				

If the AIOX is configured to a different type, the return value will be 65535 or 0xFFFF. If there is a valid analog input measurement, the register contains the current AI measurement result in Volts*100. So 537 stands for 5,37V.

But you can also read the AIOX value in percent:

AIOX:VOLTAGE INPUTS						
VOLTAGE INPUT1 IN PERCENT	3x40033 4x40033 I:40032	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of VIx:65535=N/V				
Current value of voltage input in x*100%, range 0-100% =65535,0xFFFF: The channel is not configured as voltage input						
VOLTAGE INPUT3 IN PERCENT	3x40034 4x40034 I:40033	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of VIx:65535=N/V				
VOLTAGE INPUT3 IN PERCENT	3x40035 4x40035 I:40034	0,0x0000 B:00 00			UINT16 R/O	
		Actual value of VIx:0=0,00%				

8.3.4.4.4 Howto set an analog output 0-10V or 2-10V

If an AIOX is configured either to ANALOG OUTPUT with 0-10V or to ANALOG OUTPUT with 2-10V you can use the ASCII commands SVOSV or SVOVx to set a new value for the analog output. For analog outputs configured 0-10V you can set a value between 0 to 11V. For outputs configured to 2-10V, you can set 0V to output 0V and 2-11V to output 2-11V. But every value >0 and <2V will be outputted as 2V.

VOLTAGE OUTPUTS				
SET VOLTAGE OUTPUTS IN VOLT	ASCII WRITE COMMAND	#SVOSV:<IO1VoltDbl>,<IO2VoltDbl>,<IO3VoltDbl>,<IO4VoltDbl>,<IO5VoltDbl>,<IO6VoltDbl>,<IO7VoltDbl>,<IO8VoltDbl>,<IO9VoltDbl>,<IO10VoltDbl>,<IO11VoltDbl>,<IO12VoltDbl>,<IO13VoltDbl>,<IO14VoltDbl>,<IO15VoltDbl>,<IO16VoltDbl><CR> Result: #OK<CR>	ASCII	NO
	IO1Volt	10,000		
	IO2Volt	7,500		
	IO3Volt	5,500		
	IO4Volt	2,500		
	IO5Volt	2,000		
	IO6Volt	2,000		
	IO7Volt	2,000		
	IO8Volt	2,000		
	IO9Volt	2,000		
	IO10Volt	2,000		
	IO11Volt	2,000		
	IO12Volt	2,000		
	IO13Volt	2,000		
	IO14Volt	2,000		
	IO15Volt	2,000		
	IO16Volt	2,000		
	TX	#255,SVOSV:10,7.5,5.5,2.5,2,2,2,2,2,2,2,2,2,2,2,2<CR>		
	RX	N/A		
This command sets for all VOLTAGE OUTPUT IOs the current output voltage in Volt. The range is 0.0 to 11.00V.				
SET VOLTAGE OUTPUTx IN VOLT	ASCII WRITE COMMAND	#SVOV<IONR>:<IOxVoltDbl><CR> Result: #OK<CR>	ASCII	NO
	IONR	1		
	IOxVolt	2,000		
	TX	#255,SVOV1:2<CR>		
	RX	N/A		
This command sets for VOLTAGE OUTPUT IO <IONR> the current output voltage in Volt. The range is 0.0 to 11.00V.				

But you can also set a new value for the analog outputs in percent:

SET VOLTAGE OUTPUTS IN PERCENT	ASCII WRITE COMMAND	#SVOSP:<IO1PercentDbl>,<IO2PercentDbl>,<IO3PercentDbl>,<IO4PercentDbl>,<IO5PercentDbl>,<IO6PercentDbl>,<IO7PercentDbl>,<IO8PercentDbl>,<IO9PercentDbl>,<IO10PercentDbl>,<IO11PercentDbl>,<IO12PercentDbl>,<IO13PercentDbl>,<IO14PercentDbl>,<IO15PercentDbl>,<IO16PercentDbl><CR> Result: #OK<CR>	ASCII	NO
	IO1Percent	110,000		
	IO2Percent	100,000		
	IO3Percent	75,000		
	IO4Percent	50,000		
	IO5Percent	110,000		
	IO6Percent	100,000		
	IO7Percent	75,000		
	IO8Percent	50,000		
	IO9Percent	110,000		
	IO10Percent	100,000		
	IO11Percent	75,000		
	IO12Percent	50,000		
	IO13Percent	110,000		
	IO14Percent	100,000		
	IO15Percent	75,000		
	IO16Percent	50,000		
	TX	#255,SVOSP:110,100,75,50,110,100,75,50,110,100,75,50,110,100,75,50<CR>		
	RX	N/A		
This command sets for all VOLTAGE OUTPUT IOs the current output voltage in Percent. The range is 0.0V -> 0.00% to 11.00V -> 110.00%.				
SET VOLTAGE OUTPUTx IN PERCENT	ASCII WRITE COMMAND	#SVOP<IONR>:<IOxPercentDbl><CR> Result: #OK<CR>	ASCII	NO
	IONR	1		
	IOxPercent	2,000		
	TX	#255,SVOP1:2<CR>		
	RX	N/A		
This command sets for VOLTAGE OUTPUT IO <IONR> the current output voltage in Percent. The range is 0.0V -> 0.00% to 11.00V -> 110.00%.				

To read back the actual value of the analog outputs use the commands GVOVS or GVOVx:

GET VOLTAGE OUTPUTS IN VOLT	ASCII READ COMMAND	#GVOSV<CR> Result: #GVOSV:<IO1VoltDbl>,<IO2VoltDbl>,...,<IO16VoltDbl><CR>	ASCII	
	TX	#255,GVOSV<CR>		
	RX	#1,GVOSV:0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00<CR>		
		Actual voltage output on IO1:0.00V		
		Actual voltage output on IO2:0.00V		
		Actual voltage output on IO3:0.00V		
		Actual voltage output on IO4:0.00V		
		Actual voltage output on IO5:0.00V		
		Actual voltage output on IO6:0.00V		
		Actual voltage output on IO7:0.00V		
		Actual voltage output on IO8:0.00V		
		Actual voltage output on IO9:0.00V		
		Actual voltage output on IO10:0.00V		
		Actual voltage output on IO11:0.00V		
		Actual voltage output on IO12:0.00V		
		Actual voltage output on IO13:0.00V		
		Actual voltage output on IO14:0.00V		
		Actual voltage output on IO15:0.00V		
		Actual voltage output on IO16:0.00V		
This command shows for all VOLTAGE OUTPUT IOs the current output voltage in Volt. The range is 0.0V to 11.00V. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET VOLTAGE OUTPUT IN VOLT	ASCII READ COMMAND	#GVOV<IONR><CR> Result: #GVOV<IONR>:<IOxVoltDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GVOV3<CR>		
	RX	#1,GVOV3:0.00<CR>		
		Actual voltage output on IO3:0.00V		
This command shows for VOLTAGE OUTPUT IO <IONR> the current output voltage in Volt. The range is 0.0V to 11.00V. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

Again you can read back the analog outputs also as a percent value with these commands:

GET VOLTAGE OUTPUTS IN PERCENT	ASCII READ COMMAND	#GVOSP<CR> Result: #GVOSP:<IO1PercentDbl>,<IO2PercentDbl>,...,<IO16PercentDbl><CR>	ASCII	
	TX	#255,GVOSP<CR>		
	RX	#1,GVOSP:0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00<CR>		
		Actual percentage voltage output on IO1:0.00%		
		Actual percentage voltage output on IO2:0.00%		
		Actual percentage voltage output on IO3:0.00%		
		Actual percentage voltage output on IO4:0.00%		
		Actual percentage voltage output on IO5:0.00%		
		Actual percentage voltage output on IO6:0.00%		
		Actual percentage voltage output on IO7:0.00%		
		Actual percentage voltage output on IO8:0.00%		
		Actual percentage voltage output on IO9:0.00%		
		Actual percentage voltage output on IO10:0.00%		
		Actual percentage voltage output on IO11:0.00%		
		Actual percentage voltage output on IO12:0.00%		
		Actual percentage voltage output on IO13:0.00%		
		Actual percentage voltage output on IO14:0.00%		
		Actual percentage voltage output on IO15:0.00%		
		Actual percentage voltage output on IO16:0.00%		
This command shows for all VOLTAGE OUTPUT IOs the current output voltage in Percent. The range is 0.0V -> 0.00% to 11.00V -> 110.00% (10.00V -> 100.00%). All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET VOLTAGE OUTPUT IN PERCENT	ASCII READ COMMAND	#GVOP<IONR><CR> Result: #GVOP<IONR>:<IOxPercentDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GVOP3<CR>		
	RX	#1,GVOP3:0.00<CR>		
		Actual percentage voltage output on IO3:0.00%		
This command shows for VOLTAGE OUTPUT IO <IONR> the current output voltage in Percent. The range is 0.0V -> 0.00% to 11.00V -> 110.00% (10.00V -> 100.00%). All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

In addition to detect any error in the analog output, the actual output current of the voltage output is measured too. You can read this value with the function GVOSC or GVOCx. A value >30mA is a good indication of a short cut on the analog output.

GET VOLTAGE OUTPUTS CURRENT	ASCII READ COMMAND	#GVOSC<CR> Result: #GVOSC:<IOmA1Dbl>,<IOmA2Dbl>,...,<IOmA16Dbl><CR>	ASCII	
	TX	#255,GVOSC<CR>		
	RX	#1,GVOSC:0.00,0.00,0.00,-0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00<CR>		
		Actual output current on IO1:0.00mA		
		Actual output current on IO2:0.00mA		
		Actual output current on IO3:0.00mA		
		Actual output current on IO4:-0.00mA		
		Actual output current on IO5:0.00mA		
		Actual output current on IO6:0.00mA		
		Actual output current on IO7:0.00mA		
		Actual output current on IO8:0.00mA		
		Actual output current on IO9:0.00mA		
		Actual output current on IO10:0.00mA		
		Actual output current on IO11:0.00mA		
		Actual output current on IO12:0.00mA		
		Actual output current on IO13:0.00mA		
		Actual output current on IO14:0.00mA		
		Actual output current on IO15:0.00mA		
		Actual output current on IO16:0.00mA		
This command shows for all VOLTAGE OUTPUT IOs the actual current in mA. The measurement range is 0.0mA to 35mA. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET VOLTAGE OUTPUT CURRENT	ASCII READ COMMAND	#GVOC<IONR><CR> Result: #GVOC<IONR>:<IOxmADbl><CR>	ASCII	
	IONR	1		
	TX	#255,GVOC1<CR>		
	RX	#1,GVOC1:0.00<CR>		
		Actual output current on IO1:0.00mA		
This command shows for VOLTAGE OUTPUT IO <IONR> the actual current in mA. The measurement range is 0.0mA to 35mA. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

On the MODBUS you have again holding registers to output a new value for the configured analog outputs:

AIOX:VOLTAGE OUTPUTS						
VOLTAGE OUTPUT1 IN VOLTS	3x40049 4x40049 I:40048	65535,0xFFFF B:FF FF	1100	11	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		
Current value of voltage output in x*100V, range 0-11V =65535,0xFFFF: The channel is not configured as voltage output Writing a new value onto this register sets voltage output x to a new output value in Volt						
VOLTAGE OUTPUT2 IN VOLTS	3x40050 4x40050 I:40049	65535,0xFFFF B:FF FF	1100	11	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		
VOLTAGE OUTPUT3 IN VOLTS	3x40051 4x40051 I:40050	65535,0xFFFF B:FF FF	1100	11	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		
VOLTAGE OUTPUT4 IN VOLTS	3x40052 4x40052 I:40051	65535,0xFFFF B:FF FF	1100	11	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		

You can choose also the registers to use percent values:

AIOX:VOLTAGE OUTPUTS						
VOLTAGE OUTPUT1 IN PERCENT	3x40065 4x40065 I:40064	65535,0xFFFF B:FF FF	11000	110	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		
Current value of voltage output in x*100%, range 0-110% (100%=10V) =65535,0xFFFF: The channel is not configured as voltage output Writing a new value onto this register sets voltage output x to a new output value in percent						
VOLTAGE OUTPUT2 IN PERCENT	3x40066 4x40066 I:40065	65535,0xFFFF B:FF FF	5000	50	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		
VOLTAGE OUTPUT3 IN PERCENT	3x40067 4x40067 I:40066	65535,0xFFFF B:FF FF	3000	30	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		
VOLTAGE OUTPUT4 IN PERCENT	3x40068 4x40068 I:40067	65535,0xFFFF B:FF FF	7500	75	UINT16 R/W	NO
		Actual value of VOx:65535=N/V		ENTER NEW VALUE FOR VOx		

To read back the actual output current with MODBUS registers use this registers:

AIOX:VOLTAGE OUTPUTS						
VOLTAGE OUTPUT1 MEASURED CURRENT	3x40081 4x40081 I:40080	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured output current of VOx:-32768=N/V				
Returns the measured output current in x*100mA on voltage output VOx, Range -25mA..+25mA =-32768,0x8000: The channel is not configured as voltage output						
VOLTAGE OUTPUT2 MEASURED CURRENT	3x40082 4x40082 I:40081	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured output current of VOx:-32768=N/V				
VOLTAGE OUTPUT3 MEASURED CURRENT	3x40083 4x40083 I:40082	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured output current of VOx:-32768=N/V				
VOLTAGE OUTPUT4 MEASURED CURRENT	3x40084 4x40084 I:40083	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured output current of VOx:-32768=N/V				

8.3.4.4.5 Howto read analog inputs 0-20mA or 4-20mA

If an AIOX is configured either to ANALOG INPUT with 0-20mA or to ANALOG INPUT with 4-20mA you can use the ASCII commands GCISMA or GCIMAx to read the actual value of the analog input:

CURRENT INPUTS			
GET CURRENT INPUTS IN mA	ASCII READ COMMAND	#GCISMA<CR> Result: #GCISMA:<IO1mADbl>,<IO2mADbl>,...,<IO16mADbl><CR>	ASCII
	TX	#255,GCISMA<CR>	
	RX	#1,GCISMA:999.99,999.99<CR>	
		Actual current input on IO1:999.99mA	
		Actual current input on IO2:999.99mA	
		Actual current input on IO3:999.99mA	
		Actual current input on IO4:999.99mA	
		Actual current input on IO5:999.99mA	
		Actual current input on IO6:999.99mA	
		Actual current input on IO7:999.99mA	
		Actual current input on IO8:999.99mA	
		Actual current input on IO9:999.99mA	
		Actual current input on IO10:999.99mA	
		Actual current input on IO11:999.99mA	
		Actual current input on IO12:999.99mA	
		Actual current input on IO13:999.99mA	
		Actual current input on IO14:999.99mA	
		Actual current input on IO15:999.99mA	
		Actual current input on IO16:999.99mA	
This command shows for all CURRENT INPUT IOs the current measured input current in mA. The range is 0.00 to 25.00mA All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.			
GET CURRENT INPUT IN mA	ASCII READ COMMAND	#GCIMA<IONR><CR> Result: #GCIMA<IONR>:<IOxmADbl><CR>	ASCII
	IONR	3	
	TX	#255,GCIMA3<CR>	
	RX	#1,GCIMA3:999.99<CR>	
		Actual current input on IO3:999.99mA	
This command shows for CURRENT INPUT IO <IONR> the current measured input current in mA. The range is 0.00 to 25.00mA All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.			

But you can also read this value as a percent value:

GET CURRENT INPUTS IN PERCENT	ASCII READ COMMAND	#GCISP<CR> Result: #GCISP:<IO1PercentDbl>,<IO2PercentDbl>,...,<IO16PercentDbl><CR>	ASCII
	TX	#255,GCISP<CR>	
	RX	#255,GCISP:999.99,999.99<CR>	
		Actual percentage for current input on IO1:999.99%	
		Actual percentage for current input on IO2:999.99%	
		Actual percentage for current input on IO3:999.99%	
		Actual percentage for current input on IO4:999.99%	
		Actual percentage for current input on IO5:999.99%	
		Actual percentage for current input on IO6:999.99%	
		Actual percentage for current input on IO7:999.99%	
		Actual percentage for current input on IO8:999.99%	
		Actual percentage for current input on IO9:999.99%	
		Actual percentage for current input on IO10:999.99%	
		Actual percentage for current input on IO11:999.99%	
		Actual percentage for current input on IO12:999.99%	
		Actual percentage for current input on IO13:999.99%	
		Actual percentage for current input on IO14:999.99%	
		Actual percentage for current input on IO15:999.99%	

		Actual percentage for current input on IO16:999.99%		
This command shows for all CURRENT INPUT IOs the current measured input current in Percent. The range is 0.00mA -> 0.00% to 25.00mA -> 125.00% (20mA=100%) All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET CURRENT INPUT IN PERCENT	ASCII READ COMMAND	#GCIP<IONR><CR> Result: #GCIP<IONR>:<IOxPercentDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GCIP3<CR>		
	RX	#255,GCIP3:999.99<CR>		
		Actual percentage for current input on IO3:999.99%		
This command shows for CURRENT INPUT IO <IONR> the current measured input current in Percent. The range is 0.00mA -> 0.00% to 25.00mA -> 125.00% (20mA=100%) All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

On the MODBUS side, you can read out the actual values for a current input from the registers:

AIOX:CURRENT INPUTS						
CURRENT INPUT1 IN MILLIAMPERE	3x40097 4x40097 I:40096	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of CIx:65535=N/V				
Current value of current input in x*100mA, range 0-25mA =65535,0xFFFF: The channel is not configured as current input						
CURRENT INPUT2 IN MILLIAMPERE	3x40098 4x40098 I:40097	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of VIx:65535=N/V				
CURRENT INPUT3 IN MILLIAMPERE	3x40099 4x40099 I:40098	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of VIx:65535=N/V				

or again in percent, if you like:

AIOX:CURRENT INPUTS						
CURRENT INPUT1 IN PERCENT	3x40113 4x40113 I:40112	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual value of CIx:65535=N/V				
Current value of current input in x*100%, range 0-125% (100%=20mA) =65535,0xFFFF: The channel is not configured as current input						

8.3.4.4.6 Howto set analog outputs 0-20mA or 4-20mA

If an AIOX is configured either to ANALOG OUTPUT with 0-20mA or to ANALOG OUTPUT with 4-20mA you can use the ASCII commands SCOSMA or SCOMAx to write a new value to the AIOX:

CURRENT OUTPUTS				
SET CURRENT OUTPUTS IN mA	ASCII WRITE COMMAND	#SCOSMA:<IO1mADbl>,<IO2mADbl>,<IO3mADbl>,<IO4mADbl>,<IO5mADbl>,<IO6mADbl>,<IO7mADbl>,<IO8mADbl>,<IO9mADbl>,<IO10mADbl>,<IO11mADbl>,<IO12mADbl>,<IO13mADbl>,<IO14mADbl>,<IO15mADbl>,<IO16mADbl><CR> Result: #OK<CR>	ASCII	NO
	IO1mA	2,000		
	IO2mA	4,000		
	IO3mA	6,000		
	IO4mA	25,000		
	IO5mA	,000		
	IO6mA	,000		
	IO7mA	,000		
	IO8mA	,000		
	IO9mA	,000		
	IO10mA	,000		
	IO11mA	,000		
	IO12mA	,000		
	IO13mA	,000		
	IO14mA	,000		
	IO15mA	,000		
	IO16mA	,000		
	TX	#255,SCOSMA:2,4,6,25,0,0,0,0,0,0,0,0,0,0,0,0<CR>		
	RX	N/A		
This command sets for all CURRENT OUTPUT IOs the actual output current in mA. The range is 0.00mA to 25.00mA				
SET CURRENT OUTPUTx IN mA	ASCII WRITE COMMAND	#SCOMA<IONR>:<IOxmADbl><CR> Result: #OK<CR>	ASCII	NO
	IONR	1		
	IOxVolt	2,000		
	TX	#255,SCOMA1:<IOxmADbl><CR>		
	RX	N/A		
This command sets for CURRENT OUTPUT <IONR> IOs the actual output current in mA. The range is 0.00mA to 25.00mA				

The following MODBUS registers do the same job:

AIOX:CURRENT OUTPUTS						
CURRENT OUTPUT1 IN MILIAMPERE	3x40129 4x40129 I:40128	65535,0xFFFF B:FF FF	500	5	UINT16 R/W	NO
		Actual value of COx:65535=N/V		ENTER NEW VALUE FOR COx		
Current value of current output in x*100mA, range 0-25mA =65535,0xFFFF: The channel is not configured as current output Writing a new value onto this register sets current output x to a new output value in Milliampere						
CURRENT OUTPUT2 IN MILIAMPERE	3x40130 4x40130 I:40129	65535,0xFFFF B:FF FF	500	5	UINT16 R/W	NO
		Actual value of COx:65535=N/V		ENTER NEW VALUE FOR COx		
CURRENT OUTPUT3 IN MILIAMPERE	3x40131 4x40131 I:40130	65535,0xFFFF B:FF FF	500	5	UINT16 R/W	NO
		Actual value of COx:65535=N/V		ENTER NEW VALUE FOR COx		

You can also write a new current value in percent to the AIOX with the commands SCOSP or SCOPx:

SET CURRENT OUTPUTS IN PERCENT	ASCII WRITE COMMAND	#SCOSP:<IO1PercentDbl>,<IO2PercentDbl>,<IO3PercentDbl>,<IO4PercentDbl>,<IO5PercentDbl>,<IO6PercentDbl>,<IO7PercentDbl>,<IO8PercentDbl>,<IO9PercentDbl>,<IO10PercentDbl>,<IO11PercentDbl>,<IO12PercentDbl>,<IO13PercentDbl>,<IO14PercentDbl>,<IO15PercentDbl>,<IO16PercentDbl><CR> Result: #OK<CR>	ASCII	NO
	IO1Percent	125,000		
	IO2Percent	100,000		
	IO3Percent	75,000		
	IO4Percent	50,000		
	IO5Percent	,000		
	IO6Percent	,000		
	IO7Percent	,000		
	IO8Percent	,000		
	IO9Percent	,000		
	IO10Percent	,000		
	IO11Percent	,000		
	IO12Percent	,000		
	IO13Percent	,000		
	IO14Percent	,000		
	IO15Percent	,000		
	IO16Percent	,000		
	TX	#255,SCOSP:125,100,75,50,0,0,0,0,0,0,0,0,0,0,0,0<CR>		
	RX	N/A		
This command sets for all CURRENT OUTPUT IOs the new output current in Percent. The range is 0.00mA -> 0.00% to 25.00mA -> 125.00% (20mA -> 100.00%)				
SET CURRENT OUTPUTx IN PERCENT	ASCII WRITE COMMAND	#SCOP<IONR>:<IOxPercentDbl><CR> Result: #OK<CR>	ASCII	NO
	IONR	1		
	IOxPercent	,000		
	TX	#255,SCOP1:0<CR>		
	RX	N/A		
This command sets for CURRENT OUTPUT IO <IONR> the new output current in Percent. The range is 0.00mA -> 0.00% to 25.00mA -> 125.00% (20mA -> 100.00%)				

To read back the actual value of the current output in Milliampere use the commands GCOSMA or GCOMAx:

GET CURRENT OUTPUTS IN mA	ASCII READ COMMAND	#GCOSMA<CR> Result: #GCOSMA:<IO1mADbl>,<IO2mADbl>,....,<IO16mADbl><CR>	ASCII	
	TX	#255,GCOSMA<CR>		
	RX	#255,GCOSMA:999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Actual value of current output on IO1:999.99mA		
		Actual value of current output on IO2:999.99mA		

		Actual value of current output on IO3:999.99mA		
		Actual value of current output on IO4:999.99mA		
		Actual value of current output on IO5:999.99mA		
		Actual value of current output on IO6:999.99mA		
		Actual value of current output on IO7:999.99mA		
		Actual value of current output on IO8:999.99mA		
		Actual value of current output on IO9:999.99mA		
		Actual value of current output on IO10:999.99mA		
		Actual value of current output on IO11:999.99mA		
		Actual value of current output on IO12:999.99mA		
		Actual value of current output on IO13:999.99mA		
		Actual value of current output on IO14:999.99mA		
		Actual value of current output on IO15:999.99mA		
		Actual value of current output on IO16:999.99mA		
This command shows for all CURRENT OUTPUT IOs the actual output current in mA. The range is 0.00mA to 25.00mA All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET CURRENT OUTPUT IN mA	ASCII READ COMMAND	#GCOMA<IONR><CR> Result: #GCOMA<IONR>:<IOxmADbl><CR>	ASCII	
	IONR	3		
	TX	#255,GCOMA3<CR>		
	RX	#255,GCOMA3:999.99<CR>		
		Actual value of current output on IO3:999.99mA		
This command shows for CURRENT OUTPUT IO <IONR> the actual output current in mA. The range is 0.00mA to 25.00mA All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

Use these MODBUS registers to write percentage values to the current outputs:

AIOX:CURRENT OUTPUTS						
CURRENT OUTPUT1 IN PERCENT	3x40145 4x40145 I:40144	65535,0xFFFF B:FF FF	5000	50	UINT16 R/W	NO
		Actual value of COx:65535=N/V		ENTER NEW VALUE FOR COx		
Current value of current output in x*100%, range 0-125% (100%=20mA) =65535,0xFFFF: The channel is not configured as current output Writing a new value onto this register sets current output x to a new output value in percent						
CURRENT OUTPUT2 IN PERCENT	3x40146 4x40146 I:40145	65535,0xFFFF B:FF FF	5000	50	UINT16 R/W	NO
		Actual value of COx:65535=N/V		ENTER NEW VALUE FOR COx		
CURRENT OUTPUT3 IN PERCENT	3x40147 4x40147 I:40146	65535,0xFFFF B:FF FF	5000	50	UINT16 R/W	NO
		Actual value of COx:65535=N/V		ENTER NEW VALUE FOR COx		

If you use an AIOX as current output the system measures also the actual voltage on the current outputs. You can use this value to detect some errors in the current output. Use the ASCII commands GCOSV or GCOVx to read-back the actual voltage:

GET CURRENT OUTPUTS VOLTAGE	ASCII READ COMMAND	#GCOSV<CR> Result: #GCOSV:<IO1VoltsDbl>,<IO2VoltsDbl>,...,<IO16VoltsDbl><CR>	ASCII	
	TX	#255,GCOSV<CR>		
	RX	#255,GCOSV:999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Measured voltage of current output on IO1:999.99V		
		Measured voltage of current output on IO2:999.99V		
		Measured voltage of current output on IO3:999.99V		
		Measured voltage of current output on IO4:999.99V		
		Measured voltage of current output on IO5:999.99V		
		Measured voltage of current output on IO6:999.99V		
		Measured voltage of current output on IO7:999.99V		
		Measured voltage of current output on IO8:999.99V		
		Measured voltage of current output on IO9:999.99V		
		Measured voltage of current output on IO10:999.99V		
		Measured voltage of current output on IO11:999.99V		
		Measured voltage of current output on IO12:999.99V		
		Measured voltage of current output on IO13:999.99V		
		Measured voltage of current output on IO14:999.99V		
		Measured voltage of current output on IO15:999.99V		
		Measured voltage of current output on IO16:999.99V		
This command shows for all CURRENT OUTPUT IOs the actual output voltage in Volt. The range is 0-10V All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET CURRENT OUTPUT VOLTAGE	ASCII READ COMMAND	#GCOV<IONR><CR> Result: #GCOV<IONR>:<IOxVoltDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GCOV3<CR>		
	RX	#255,GCOV3:999.99<CR>		
		Measured voltage of current output on IO3:999.99V		
This command shows for CURRENT OUTPUT IO <IONR> the actual output voltage in Volt. The range is 0-10V All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

This MODBUS registers have the same purpose to give back the actual measured voltage for a current output.

AIOX:CURRENT OUTPUTS						
CURRENT OUTPUT1 MEASURED VOLTS	3x40161 4x40161 I:40160	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured output voltage COx:65535=N/V				
Current measured output voltage for current output x*100V, range 0-10V =65535,0xFFFF: The channel is not configured as current output						
CURRENT OUTPUT2 MEASURED VOLTS	3x40162 4x40162 I:40161	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured output voltage COx:65535=N/V				
CURRENT OUTPUT3 MEASURED VOLTS	3x40163 4x40163 I:40162	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured output voltage COx:65535=N/V				

Or you choose to read-back the values in percent with GCOSP or GCOPx:

GET CURRENT OUTPUTS IN PERCENT	ASCII READ COMMAND	#GCOSP<CR> Result: #GCOSP:<IO1PercentDbl>,<IO2PercentDbl>,....,<IO16PercentDbl><CR>	ASCII	
	TX	#255,GCOSP<CR>		
	RX	#255,GCOSP:999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Actual percentage of current output on IO1:999.99%		
		Actual percentage of current output on IO2:999.99%		
		Actual percentage of current output on IO3:999.99%		
		Actual percentage of current output on IO4:999.99%		
		Actual percentage of current output on IO5:999.99%		
		Actual percentage of current output on IO6:999.99%		
		Actual percentage of current output on IO7:999.99%		
		Actual percentage of current output on IO8:999.99%		
		Actual percentage of current output on IO9:999.99%		
		Actual percentage of current output on IO10:999.99%		
		Actual percentage of current output on IO11:999.99%		
		Actual percentage of current output on IO12:999.99%		
		Actual percentage of current output on IO13:999.99%		
		Actual percentage of current output on IO14:999.99%		
		Actual percentage of current output on IO15:999.99%		
		Actual percentage of current output on IO16:999.99%		
This command shows for all CURRENT OUTPUT IOs the actual output current in Percent. The range is 0.00mA -> 0.00% to 25.00mA -> 125.00% (20mA -> 100.00%) All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET CURRENT OUTPUT IN PERCENT	ASCII READ COMMAND	#GCOP<IONR><CR> Result: #GCOP<IONR>:<IOxPercentDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GCOP3<CR>		
	RX	#255,GCOP3:999.99<CR>		
		Actual percentage of current output on IO3:999.99%		
This command shows for CURRENT OUTPUT IO <IONR> the actual output current in Percent. The range is 0.00mA -> 0.00% to 25.00mA -> 125.00% (20mA -> 100.00%) All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

8.3.4.4.7 Howto read a digital input

If an AIOX is configured either to DIGITAL INPUT for 24Vdc, logic or to DIGITAL INPUT for 24Vdc, loop powered you can read with the ASCII commands GVDIS or GVDIx the current status of the digital inputs:

VOLTAGE DIGITAL INPUTS				
GET VOLTAGE DIGITAL INPUTS	ASCII READ COMMAND	#GVDIS<CR> Result: #GVDIS:<IODI1Dec>,<IODI2Decl>,....,<IODI16Dec><CR>	ASCII	
	TX	#255,GVDIS<CR>		
	RX	#255,GVDIS:X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X<CR>		
		Actual voltage digital input state on IO1:X		
		Actual voltage digital input state on IO2:X		
		Actual voltage digital input state on IO3:X		
		Actual voltage digital input state on IO4:X		
		Actual voltage digital input state on IO5:X		
		Actual voltage digital input state on IO6:X		
		Actual voltage digital input state on IO7:X		
		Actual voltage digital input state on IO8:X		
		Actual voltage digital input state on IO9:X		
		Actual voltage digital input state on IO10:X		
		Actual voltage digital input state on IO11:X		
		Actual voltage digital input state on IO12:X		
		Actual voltage digital input state on IO13:X		
		Actual voltage digital input state on IO14:X		
		Actual voltage digital input state on IO15:X		
		Actual voltage digital input state on IO16:X		
This command shows for all VOLTAGE DIGITAL INPUT IOs the current state. The digital input can have the values 0 and 1. All IOs with a different usage type will return X to indicate, that no measurement is done.				
GET VOLTAGE DIGITAL INPUT	ASCII READ COMMAND	#GVDI<IONR><CR> Result: #GVDI<IONR>:<IOxDIDec><CR>	ASCII	
	IONR	3		
	TX	#255,GVDI3<CR>		
	RX	#255,GVDI3:X<CR>		
		Actual voltage digital input state on IO3:X		
This command shows for VOLTAGE DIGITAL INPUT IO <IONR> the current state. The digital input can have the values 0 and 1. All IOs with a different usage type will return X to indicate, that no measurement is done.				

If you want to read the status with MODBUS use this registers:

AIOX:DIGITAL INPUTS						
DIGITAL INPUT1	3x40177 4x40177 I:40176	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual state of digital input DIx:65535=N/V				
Current measured state of digital input DIx =0: Digital input is OFF or loop is closed =1: Digital input is ON (+24V attached) or loop is open =65535,0xFFFF: The channel is not configured as digital input						
DIGITAL INPUT2	3x40178 4x40178 I:40177	1,0x0001 B:00 01			UINT16 R/O	
		Actual state of digital input DIx:1=ON				
DIGITAL INPUT3	3x40179 4x40179 I:40178	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured output voltage COx:65535=N/V				

The system measures also the actual current for the digital inputs. You can read this current with the ASCII commands GVDISC or GVDICx:

GET VOLTAGE DIGITAL INPUTS CURRENT	ASCII READ COMMAND	#GVDISC<CR> Result: #GVDISC:<IOmA1Dbl>,<IOmA2Dbl>,....,<IOmA16Dbl><CR>	ASCII	
	TX	#255,GVDISC<CR>		
	RX	#255,GVDISC:999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Actual input current on IO1:999.99mA		
		Actual input current on IO2:999.99mA		
		Actual input current on IO3:999.99mA		
		Actual input current on IO4:999.99mA		
		Actual input current on IO5:999.99mA		
		Actual input current on IO6:999.99mA		
		Actual input current on IO7:999.99mA		
		Actual input current on IO8:999.99mA		
		Actual input current on IO9:999.99mA		
		Actual input current on IO10:999.99mA		
		Actual input current on IO11:999.99mA		
		Actual input current on IO12:999.99mA		
		Actual input current on IO13:999.99mA		
		Actual input current on IO14:999.99mA		
		Actual input current on IO15:999.99mA		
		Actual input current on IO16:999.99mA		
This command shows for all VOLTAGE DIGITAL INPUT IOs the actual current in mA. The measurement range is 0.0mA to 35mA. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				
GET VOLTAGE DIGITAL INPUT CURRENT	ASCII READ COMMAND	#GVDIC<IONR><CR> Result: #GVDIC<IONR>:<IOxmADbl><CR>	ASCII	
	IONR	1		
	TX	#255,GVDIC1<CR>		
	RX	#255,GVDIC1:999.99<CR>		
		Actual input current on IO1:999.99mA		
This command shows for VOLTAGE DIGITAL INPUT IO <IONR> the actual current in mA. The measurement range is 0.0mA to 35mA. All IOs with a different usage type will return 999.99 to indicate, that no measurement is done.				

The same measured current on the MODBUS registers:

DIGITAL INPUT1 MEASURED CURRENT	3x40193 4x40193 I:40192	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured output current of DIx:-32768=N/V				
Returns the measured output current in x*100mA on DIGITAL INPUT VOx, Range -25mA..+25mA =-32768,0x8000: The channel is not configured as DIGITAL INPUT						
DIGITAL INPUT2 MEASURED CURRENT	3x40194 4x40194 I:40193	0,0x0000 B:00 00			SINT16 R/O	
		Actual measured output current of DIx:0=0,00mA				
DIGITAL INPUT3 MEASURED CURRENT	3x40195 4x40195 I:40194	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured output current of DIx:-32768=N/V				

8.3.4.4.8 Howto read a resistor value

If an AIOX is configured to RTD SENSOR INPUT, the input will measure the current resistor value and return this measurement value in Ohm.
Therefore you have the ASCII commands GRTDISOHM and GRTDIOHMx. Both will return the actual resistor value:

RTD INPUTS				
GET RTD INPUTS IN OHM	ASCII READ COMMAND	#GRTDISOHM<CR> Result: #GRTDISOHM:<IO1OhmDbl>,<IO2OhmDbl>,...,<IO16OhmDbl><CR>	ASCII	
	TX	#255,GRTDISOHM<CR>		
	RX	#255,GRTDISOHM:99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999<CR>		
		Actual measured RTD input on IO1:99999999.999Ohm		
		Actual measured RTD input on IO2:99999999.999Ohm		
		Actual measured RTD input on IO3:99999999.999Ohm		
		Actual measured RTD input on IO4:99999999.999Ohm		
		Actual measured RTD input on IO5:99999999.999Ohm		
		Actual measured RTD input on IO6:99999999.999Ohm		
		Actual measured RTD input on IO7:99999999.999Ohm		
		Actual measured RTD input on IO8:99999999.999Ohm		
		Actual measured RTD input on IO9:99999999.999Ohm		
		Actual measured RTD input on IO10:99999999.999Ohm		
		Actual measured RTD input on IO11:99999999.999Ohm		
		Actual measured RTD input on IO12:99999999.999Ohm		
		Actual measured RTD input on IO13:99999999.999Ohm		
		Actual measured RTD input on IO14:99999999.999Ohm		
		Actual measured RTD input on IO15:99999999.999Ohm		
		Actual measured RTD input on IO16:99999999.999Ohm		
This command shows for RTD INPUT IOs the actual measured RTD value in Ohm. The range is 0.000Ohm to 1000000.00Ohm All IOs with a different usage type will return 99999999.999 to indicate, that no measurement is done.				
GET RTD INPUT IN OHM	ASCII READ COMMAND	#GRTDIOHM<IONR><CR> Result: #GRTDIOHM<IONR>:<IOxOhmDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GRTDIOHM3<CR>		
	RX	#255,GRTDIOHM3:99999999.999<CR>		
		Actual measured RTD input on IO3:99999999.999Ohm		

In addition the AIOX has an integrated average function: It sums up 100 measurements and then updates the average values. The retrieve this average values use the ASCII commands GAVGRTDISOHM or GAVGRTDIOHMx:

GET AVERAGE RTD INPUTS IN OHM	ASCII READ COMMAND	#GAVGRTDISOHM<CR> Result: #GAVGRTDISOHM:<IO1OhmDbl>,<IO2OhmDbl>,....,<IO16OhmDbl><CR>	ASCII	
	TX	#255,GAVGRTDISOHM<CR>		
	RX	#255,GAVGRTDISOHM:99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999,99999999.999<CR>		
		Average measured RTD input on IO1:99999999.999Ohm		
		Average measured RTD input on IO2:99999999.999Ohm		
		Average measured RTD input on IO3:99999999.999Ohm		
		Average measured RTD input on IO4:99999999.999Ohm		
		Average measured RTD input on IO5:99999999.999Ohm		
		Average measured RTD input on IO6:99999999.999Ohm		
		Average measured RTD input on IO7:99999999.999Ohm		
		Average measured RTD input on IO8:99999999.999Ohm		
		Average measured RTD input on IO9:99999999.999Ohm		
		Average measured RTD input on IO10:99999999.999Ohm		
		Average measured RTD input on IO11:99999999.999Ohm		
		Average measured RTD input on IO12:99999999.999Ohm		
		Average measured RTD input on IO13:99999999.999Ohm		
		Average measured RTD input on IO14:99999999.999Ohm		
		Average measured RTD input on IO15:99999999.999Ohm		
		Average measured RTD input on IO16:99999999.999Ohm		
This command shows for RTD INPUT IOs the average measured RTD value in Ohm. The range is 0.000Ohm to 1000000.00Ohm All IOs with a different usage type will return 99999999.999 to indicate, that no measurement is done.				
GET AVG RTD INPUT IN OHM	ASCII READ COMMAND	#GAVGRTDIOHM<IONR><CR> Result: #GAVGRTDIOHM<IONR>:<IOxOhmDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GAVGRTDIOHM3<CR>		
	RX	#255,GAVGRTDIOHM3:99999999.999<CR>		
		Average measured RTD input on IO3:99999999.999Ohm		
This command shows for RTD INPUT IO <IONR> the average measured RTD value in Ohm. The range is 0.000Ohm to 1000000.00Ohm All IOs with a different usage type will return 99999999.999 to indicate, that no measurement is done.				

You can read the current measurement also via MODBUS with the holding registers. We have mirrored the values in various register banks to ease the use of the resistor value. The first bank holds the current resistor value multiplied by 10. But with this registers you can only measure 0 to 60000Ohm with the accuracy of 0.1Ohm!

AIOX:RTD INPUTS						
RTD INPUT1 IN OHM*10	3x41001 4x41001 I:41000	65534,0xFFFE B:FF FE			UINT16 R/O	
		Actual measured ohm value of RTDIx:65534=OPEN				
Current measured RTD in Ohm*10 between 0 and 600000 =0..60000: Current measured resistance in Ohm*10 =65534,0xFFFE: The sensor or cabling is open (broken, not connected, or out of range) =65535,0xFFFF: The channel is not configured as RTD input						
RTD INPUT2 IN OHM*10	3x41002 4x41002 I:41001	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured ohm value of RTDIx:65535=N/V				
RTD INPUT3 IN OHM*10	3x41003 4x41003 I:41002	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured ohm value of RTDIx:65535=N/V				

Again you can also read the average values from this holding registers:

AIOX:AVERAGE RTD INPUTS						
AVERAGE RTD INPUT1 IN OHM*10	3x42001 4x42001 I:42000	65534,0xFFFE B:FF FE			UINT16 R/O	
		Measured average ohm value of RTDIx:65534=OPEN				
Measured average RTD in Ohm*10 between 0 and 600000 =0..60000: Measured average resistance in Ohm*10 =65534,0xFFFE: The sensor or cabling is open (broken, not connected, or out of range) =65535,0xFFFF: The channel is not configured as RTD input						
AVERAGE RTD INPUT2 IN OHM*10	3x42002 4x42002 I:42001	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average ohm value of RTDIx:65535=N/V				
AVERAGE RTD INPUT3 IN OHM*10	3x42003 4x42003 I:42002	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average ohm value of RTDIx:65535=N/V				

To be more accurate but with a smaller range you can read holding registers to retrieve the current resistor value between 0 and 60000Ohm with the accuracy of 1Ohm:

AIOX:RTD INPUTS						
RTD INPUT1 IN OHM	3x41017 4x41017 I:41016	8822,0x2276 B:22 76			UINT16 R/O	
		Actual measured ohm value of RTDIx:8822=8822Ohm				
Current measured RTD in Ohm*1 between 0 and 60000 =0..60000: Current measured resistance in Ohm*1 =65534,0xFFFE: The sensor or cabling is open (broken, not connected, or out of range) =65535,0xFFFF: The channel is not configured as RTD input						
RTD INPUT2 IN OHM	3x41018 4x41018 I:41017	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured ohm value of RTDIx:65535=N/V				
RTD INPUT3 IN OHM	3x41019 4x41019 I:41018	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured ohm value of RTDIx:65535=N/V				

Again as average value in this registers:

AIOX:AVERAGE RTD INPUTS						
AVERAGE RTD INPUT1 IN OHM	3x42017 4x42017 I:42016	8788,0x2254 B:22 54			UINT16 R/O	
		Measured average ohm value of RTDIx:8788=8788Ohm				
Measured average RTD in Ohm*1 between 0 and 60000 =0..60000: Measured average resistance in Ohm*1 =65534,0xFFFE: The sensor or cabling is open (broken, not connected, or out of range) =65535,0xFFFF: The channel is not configured as RTD input						
AVERAGE RTD INPUT2 IN OHM	3x42018 4x42018 I:42017	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average ohm value of RTDIx:65535=N/V				
AVERAGE RTD INPUT3 IN OHM	3x42019 4x42019 I:42018	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average ohm value of RTDIx:65535=N/V				

Another register bank stores higher resistor values between 0 and 600000Ohm with the accuracy of 10Ohm.

AIOX:RTD INPUTS						
RTD INPUT1 IN OHM/10	3x41033 4x41033 I:41032	882,0x0372 B:03 72			UINT16 R/O	
		Actual measured ohm value of RTDIx:882=8820Ohm				
Current measured RTD in Ohm/10 between 0 and 60000 =0..60000: Current measured resistance in Ohm/10 =65534,0xFFFE: The sensor or cabling is open (broken, not connected, or out of range) =65535,0xFFFF: The channel is not configured as RTD input						
RTD INPUT2 IN OHM/10	3x41034 4x41034 I:41033	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured ohm value of RTDIx:65535=N/V				
RTD INPUT3 IN OHM/10	3x41035 4x41035 I:41034	65535,0xFFFF B:FF FF			UINT16 R/O	
		Actual measured ohm value of RTDIx:65535=N/V				

Again also as average values:

AIOX:AVERAGE RTD INPUTS						
AVERAGE RTD INPUT1 IN OHM/10	3x42033 4x42033 I:42032	879,0x036F B:03 6F			UINT16 R/O	
		Measured average ohm value of RTDIx:879=8790Ohm				
Measured average RTD in Ohm/10 between 0 and 60000 =0..60000: Measured average resistance in Ohm/10 =65534,0xFFFE: The sensor or cabling is open (broken, not connected, or out of range) =65535,0xFFFF: The channel is not configured as RTD input						
AVERAGE RTD INPUT2 IN OHM/10	3x42034 4x42034 I:42033	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average ohm value of RTDIx:65535=N/V				
AVERAGE RTD INPUT3 IN OHM/10	3x42035 4x42035 I:42034	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average ohm value of RTDIx:65535=N/V				

All those above registers are UINT16. But you can read the full resistor value from UINT32 registers in two encodings:

The next register bank hold the resistor values in Ohm*100 as values between 0 and 1MOhm with two commas in UINT32 format:

AIOX:RTD INPUTS						
RTD INPUT1 IN OHM*100	3x41501 4x41501 I:41500	883118,0x000D79AE B:00 0D 79 AE			UINT32 R/O	
		Actual measured ohm value of RTDIx:883118=8831,18Ohm				
Current measured RTD in Ohm*100 =0xFFFFFFFF: The channel is not configured as RTD input						
RTD INPUT2 IN OHM*100	3x41503 4x41503 I:41502	4294967295,0xFFFFFFFF B:FF FF FF FF			UINT32 R/O	
		Actual measured ohm value of RTDIx:-1=N/V				
RTD INPUT3 IN OHM*100	3x41505 4x41505 I:41504	4294967295,0xFFFFFFFF B:FF FF FF FF			UINT32 R/O	
		Actual measured ohm value of RTDIx:-1=N/V				

The same values are stored in another register bank in UINT32R format, because there is not only one encoding standard in MODBUS how to store 32 bit value sin 16 bit holding or input registers. See the chapter about MODBUS encoding for more information:

AIOX:RTD INPUTS						
RTD INPUT1 IN OHM*100	3x41533 4x41533 I:41532	883291,0x000D7A5B B:7A 5B 00 0D			UINT32R R/O	
		Actual measured ohm value of RTDIx:883291=8832,91Ohm				
Current measured RTD in Ohm*100 =0xFFFFFFFF: The channel is not configured as RTD input						
RTD INPUT2 IN OHM*100	3x41535 4x41535 I:41534	4294967295,0xFFFFFFFF B:FF FF FF FF			UINT32 R/O	
		Actual measured ohm value of RTDIx:-1=N/V				
RTD INPUT3 IN OHM*100	3x41537 4x41537 I:41536	4294967295,0xFFFFFFFF B:FF FF FF FF			UINT32 R/O	
		Actual measured ohm value of RTDIx:-1=N/V				

The same with two banks with the average resistor values:

AIOX:RTD INPUTS						
AVERAGE RTD INPUT1 IN OHM*100	3x42501 4x42501 I:42500	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		Measured average ohm value of RTDIx:0=0,00Ohm				
Measured average RTD in Ohm*100 =0xFFFFFFFF: The channel is not configured as RTD input						
AVERAGE RTD INPUT2 IN OHM*100	3x42503 4x42503 I:42502	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		Measured average ohm value of RTDIx:0=0,00Ohm				
AVERAGE RTD INPUT3 IN OHM*100	3x42505 4x42505 I:42504	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		Measured average ohm value of RTDIx:0=0,00Ohm				
AIOX:AVERAGE RTD INPUTS						
AVERAGE RTD INPUT1 IN OHM*100	3x42533 4x42533 I:42532	0,0x00000000 B:00 00 00 00			UINT32R R/O	
		Measured average ohm value of RTDIx:0=0,00Ohm				
Measured average RTD in Ohm*100 =0xFFFFFFFF: The channel is not configured as RTD input						
AVERAGE RTD INPUT2 IN OHM*100	3x42535 4x42535 I:42534	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		Measured average ohm value of RTDIx:0=0,00Ohm				
AVERAGE RTD INPUT3 IN OHM*100	3x42537 4x42537 I:42536	0,0x00000000 B:00 00 00 00			UINT32 R/O	
		Measured average ohm value of RTDIx:0=0,00Ohm				

8.3.4.4.9 Howto read a PT100,PT1000,NI1000-DIN43760 sensor

If an AIOX is configured to RTD SENSOR INPUT, the input will measure the current resistor value. Internally the processor converts and linearise this resistor value and check the measurement range to create the current temperature value, when you connect a PT100, PT1000 or NI1000-DIN43760 2-wire sensor to the AIOX input.

You can read out the actual temperature measurement with the following ASCII commands:

- GRTDISPT100C, GRTDIPT100Cx → Actual PT100 sensor temperature in °Celsius
- GRTDISPT100K, GRTDIPT100Kx → Actual PT100 sensor temperature in °Kelvin
- GRTDISPT100F, GRTDIPT100Fx → Actual PT100 sensor temperature in °Fahrenheit

- GRTDISPT1000C, GRTDIPT1000Cx → Actual PT1000 sensor temperature in °Celsius
- GRTDISPT1000K, GRTDIPT1000Kx → Actual PT1000 sensor temperature in °Kelvin
- GRTDISPT1000F, GRTDIPT1000Fx → Actual PT1000 sensor temperature in °Fahrenheit

- GRTDISNI1000DIN43760C, GRTDINI1000DIN43760Cx → Actual NI1000 sensor temperature with DIN43760 linearisation in °Celsius
- GRTDISPT1000DIN43760K, GRTDINI1000DIN43760Kx → Actual NI1000 sensor temperature with DIN43760 linearisation in °Kelvin
- GRTDISPT1000DIN43760F, GRTDINI1000DIN43760Fx → Actual NI1000 sensor temperature with DIN43760 linearisation in °Fahrenheit

As an example here the commands to read out a PT1000 sensor in °Celsius in ASCII:

RTD INPUTS PT1000 CELSIUS				
GET RTD INPUTS AS PT1000 CELSIUS	ASCII READ COMMAND	#GRTDISPT1000C<CR> Result: #GRTDISPT1000C:<RTD1Dbl>,<RTD2Dbl>,...,<RTD16Dbl><CR>	ASCII	
	TX	#255,GRTDISPT1000C<CR>		
	RX	#255,GRTDISPT1000C:9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990<CR>		
		Actual measured RTD input as PT1000 on IO1:9999.990°C		
		Actual measured RTD input as PT1000 on IO2:9999.990°C		
		Actual measured RTD input as PT1000 on IO3:9999.990°C		
		Actual measured RTD input as PT1000 on IO4:9999.990°C		
		Actual measured RTD input as PT1000 on IO5:9999.990°C		
		Actual measured RTD input as PT1000 on IO6:9999.990°C		
		Actual measured RTD input as PT1000 on IO7:9999.990°C		
		Actual measured RTD input as PT1000 on IO8:9999.990°C		
		Actual measured RTD input as PT1000 on IO9:9999.990°C		
		Actual measured RTD input as PT1000 on IO10:9999.990°C		
		Actual measured RTD input as PT1000 on IO11:9999.990°C		
		Actual measured RTD input as PT1000 on IO12:9999.990°C		
		Actual measured RTD input as PT1000 on IO13:9999.990°C		
		Actual measured RTD input as PT1000 on IO14:9999.990°C		
		Actual measured RTD input as PT1000 on IO15:9999.990°C		
		Actual measured RTD input as PT1000 on IO16:9999.990°C		
This command shows for RTD INPUT IOs the actual measured RTD value linearized as PT100 sensor in °Celisus. -999.990: Temperature is lower than 50°C +999.990: Temperature is higher than 130°C All IOs with a different usage type will return 9999.990 to indicate, that no measurement is done.				
GET RTD INPUT AS PT1000 CELSIUS	ASCII READ COMMAND	#GRTDIPT1000C<IONR><CR> Result: #GRTDIPT1000C<IONR>:<IOxDbl><CR>	ASCII	
	IONR	3		
	TX	#255,GRTDIPT1000C3<CR>		
	RX	#255,GRTDIPT1000C3:9999.990<CR>		
		Actual measured RTD input as PT1000 on IO3:9999.990°C		
This command shows for RTD INPUT IO <IONR> the actual measured RTD value linearized as PT100 sensor in °Celisus -999.990: Temperature is lower than 50°C +999.990: Temperature is higher than 130°C All IOs with a different usage type will return 9999.990 to indicate, that no measurement is done.				

You can read out also an average temperature with the following ASCII commands. The average is calculated from the sum of the last 100 measurements:

- GAVGRTDISPT100C, GAVGRTDIPT100Cx → Actual PT100 sensor temperature in °Celsius
- GAVGRTDISPT100K, GAVGRTDIPT100Kx → Actual PT100 sensor temperature in °Kelvin
- GAVGRTDISPT100F, GAVGRTDIPT100Fx → Actual PT100 sensor temperature in °Fahrenheit

- GAVGRTDISPT1000C, GAVGRTDIPT1000Cx → Actual PT1000 sensor temperature in °Celsius
- GAVGRTDISPT1000K, GAVGRTDIPT1000Kx → Actual PT1000 sensor temperature in °Kelvin
- GAVGRTDISPT1000F, GAVGRTDIPT1000Fx → Actual PT1000 sensor temperature in °Fahrenheit

- GAVGRTDISNI1000DIN43760C, GAVGRTDINI1000DIN43760Cx → Actual NI1000 sensor temperature with DIN43760 linearisation in °Celsius
- GAVGRTDISPT1000DIN43760K, GAVGRTDINI1000DIN43760Kx → Actual NI1000 sensor temperature with DIN43760 linearisation in °Kelvin
- GAVGRTDISPT1000DIN43760F, GAVGRTDINI1000DIN43760Fx → Actual NI1000 sensor temperature with DIN43760 linearisation in °Fahrenheit

As an example here the commands to read out the average value of a NI1000-DIN43760 sensor in °Celsius in ASCII:

GET AVERAGE RTD INPUTS AS NI1000 DIN43760 CELSIUS	ASCII READ COMMAND	#GAVGRTDISNI1000DIN43760C<CR> Result: #GAVGRTDISNI1000DIN43760C:<RTD1Dbl>,<RTD2Dbl>,...,<RTD16Dbl><CR>	ASCII	
	TX	#255,GAVGRTDISNI1000DIN43760C<CR>		
	RX	#255,GAVGRTDISNI1000DIN43760C:9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990,9999.990<CR>		
		Average measured RTD input as NI1000-DIN43760 on IO1:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO2:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO3:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO4:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO5:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO6:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO7:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO8:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO9:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO10:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO11:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO12:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO13:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO14:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO15:9999.990°C		
		Average measured RTD input as NI1000-DIN43760 on IO16:9999.990°C		
This command shows for RTD INPUT IOs the average measured RTD value linearized as PT100 sensor in °Celisus. -999.990: Temperature is lower than 50°C +999.990: Temperature is higher than 130°C All IOs with a different usage type will return 9999.990 to indicate, that no measurement is done.				
GET AVG RTD INPUT AS NI1000 DIN43760 CELSIUS	ASCII READ COMMAND	#GAVGRTDINI1000DIN43760C<IONR><CR> Result: #GAVGRTDINI1000DIN43760C<IONR>:<IOxDbl><CR>	ASCII	
	IONR	16		
	TX	#255,GAVGRTDINI1000DIN43760C16<CR>		
	RX	#255,GAVGRTDINI1000DIN43760C16:9999.990<CR>		
		Average measured RTD input as NI1000-DIN43760 on IO16:9999.990°C		
This command shows for RTD INPUT IO <IONR> the average measured RTD value linearized as PT100 sensor in °Celisus -999.990: Temperature is lower than 50°C +999.990: Temperature is higher than 130°C All IOs with a different usage type will return 9999.990 to indicate, that no measurement is done.				

But you can read the PT100, PT1000 and NI1000-DI43760 sensors also via MODBUS registers. You will find tables for reading the sensors in CELSIUS, FAHRENHEIT or KELVIN. As an example here we show the table for PT100 readout in °Celsius:

AIOX:RTD INPUTS PT100 CELSIUS						
RTD INPUT1 AS PT100 IN CELSIUS	3x41049 4x41049 I:41048	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured PT100 temperature RTDIx:-32768=N/V				
Current measured RTD sensor value linearized as PT100 sensor in Celsius*100 in the range of -5000 to +13000 for -50.0 to +130.0 °C -32766,0x8002: Measured value is below -50°C -32767,0x8001: Measured value is above +130°C -32768,0x8000: The channel is not configured as RTD input						
RTD INPUT2 AS PT100 IN CELSIUS	3x41050 4x41050 I:41049	-32768,0x8000 B:80 00			SINT16 R/O	
		Actual measured PT100 temperature RTDIx:-32768=N/V				

Again you can also read the average value for one sensor:

AIOX:AVERAGE RTD INPUTS NI1000-DIN43760 KELVIN						
AVERAGE RTD INPUT1 AS NI1000-DIN43760 IN KELVIN	3x42129 4x42129 I:42128	65535,0xFFFF B:FF FF			UINT16 R/O	
		Measured average NI1000-DIN43760 temperature RTDIx:65535=655,35°K				
Average value of measured RTD sensor linearized as NI1000-DIN43760 sensor in Kelvin*100 in the range of 22315 to 40315 for 223.15 to 403.15 °K 65533,0xFFFD: Measured value is below 223,15°K 65534,0xFFFE: Measured value is above 403,15°K 65535,0xFFFF: The channel is not configured as RTD input						

8.3.4.4.10 Howto set output values for INIT & IO WATCHDOG

If an AIOX is configured as an output, you can set a start up value, which is stored in the FRAM. This value will be used as an init value after a power-on and outputted on the voltage or current outputs. Also this value will be outputted if you have set a IO watchdog and the ASCII and MODBUS communication meet the defined watchdog timeout. So you can bring your external devices into a defined state.

Take the ASCII commands SCFGOVS to define for all analog outputs a startup value or use SCFGOVx to set a specific output value. The new value is stored in the FRAM:

SET CONFIG OUTPUT VALUES	ASCII WRITE COMMAND	#SCFGOVS:<IO1CfgValDbl>,<IO2CfgValDbl>,<IO3CfgValDbl>,<IO4CfgValDbl>,<IO5CfgValDbl>,<IO6CfgValDbl>,<IO7CfgValDbl>,<IO8CfgValDbl>,<IO9CfgValDbl>,<IO10CfgValDbl>,<IO11CfgValDbl>,<IO12CfgValDbl>,<IO13CfgValDbl>,<IO14CfgValDbl>,<IO15CfgValDbl>,<IO16CfgValDbl><CR> Result: #OK<CR>	ASCII	YES
	IO1Value	,000		
	IO2Value	,000		
	IO3Value	,000		
	IO4Value	,000		
	IO5Value	,000		
	IO6Value	,000		
	IO7Value	,000		
	IO8Value	,000		
	IO9Value	,000		
	IO10Value	,000		
	IO11Value	,000		
	IO12Value	,000		
	IO13Value	,000		
	IO14Value	,000		
	IO15Value	,000		
	IO16Value	,000		
	TX	#255,SCFGOVS:0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<CR>		
	RX	N/A		
This command sets for all outputs the standard value in Volt or in mA, which are used when the controller is restartet or performing a watchdog reset and the channel is used as voltage output or current output. For voltage outputs the range is 0 to 11,0V. For current outputs the range is 0 to 25mA.				
SET CONFIG OUTPUT VALUEx	ASCII WRITE COMMAND	#SCFGOV<IONR>:<IOxCfgValueDbl><CR> Result: #OK<CR>	ASCII	YES
	IONR	16		
	IOxCfgValue	,000		
	TX	#255,SCFGOV16:0<CR>		
	RX	N/A		
This command sets for one outputs the standard value in Volt or in mA, which is used when the controller is restartet and the channel is used as voltage output or current output. For voltage outputs the range is 0 to 11,0V. For current outputs the range is 0 to 25mA.				

You can read the current settings with GCFGOVS and GCFGOVx:

GET CONFIG OUTPUT VALUES	ASCII READ COMMAND	#GCFGOVS<CR> Result: #GCFGOVS:<IOVolt1Dbl>,<IOVolt2Dbl>,....,<IOVolt16Dbl><CR>	ASCII	
	TX	#255,GCFGOVS<CR>		
	RX	#255,GCFGOVS:0.00,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99,999.99<CR>		
		Actual config value on IO1:0.00V or mA		
		Actual config value on IO2:999.99V or mA		
		Actual config value on IO3:999.99V or mA		
		Actual config value on IO4:999.99V or mA		
		Actual config value on IO5:999.99V or mA		
		Actual config value on IO6:999.99V or mA		
		Actual config value on IO7:999.99V or mA		
		Actual config value on IO8:999.99V or mA		
		Actual config value on IO9:999.99V or mA		
		Actual config value on IO10:999.99V or mA		
		Actual config value on IO11:999.99V or mA		
		Actual config value on IO12:999.99V or mA		
		Actual config value on IO13:999.99V or mA		
		Actual config value on IO14:999.99V or mA		
		Actual config value on IO15:999.99V or mA		
		Actual config value on IO16:999.99V or mA		
This command shows for all channels the current saved startup values for use as voltage or current outputs For voltage outputs the range is 0 to 11,0V. For current outputs the range is 0 to 25mA. All IOs with a different usage type will return 999.99.				
GET CONFIG OUTPUT VALUE	ASCII READ COMMAND	#GCFGOV<IONR><CR> Result: #GCFGOV<IONR>:<IOxValueDbl><CR>	ASCII	
	IONR	1		
	TX	#255,GCFGOV1<CR>		
	RX	#255,GCFGOV1:0.00<CR>		
		Actual config value on IO1:0.00V or mA		
This command shows for one channel the current saved startup value for use as voltage or current output. For voltage outputs the range is 0 to 11,0V. For current outputs the range is 0 to 25mA. All IOs with a different usage type will return 999.99.				

But you can read and write this values also via MODBUS holding registers:

CONFIG OUTPUT VALUE AIOX1	3x44001 4x44001 I:44000	100,0x0064 B:00 64	100	1	UINT16 R/W	YES
		Actual config value for AIOx:1,00 V or mA		ENTER NEW CONFIG VALUE FOR AIOx		
This command sets for all outputs the standard value in Volt*100 or in mA*100, which are used when the controller is restartet or a watchdog condition has occured and the channel is used as voltage output or current output. For voltage outputs the range is 0 to 1100 (0 to 11,0V). For current outputs the range is 0 to 2500 (0 to 25mA). All IOs with a different usage type will return 65535,0xFFFF.						
CONFIG OUTPUT VALUE AIOX2	3x44002 4x44002 I:44001	200,0x00C8 B:00 C8	200	2	UINT16 R/W	YES
		Actual config value for AIOx:2,00 V or mA		ENTER NEW CONFIG VALUE FOR AIOx		
CONFIG OUTPUT VALUE AIOX3	3x44003 4x44003 I:44002	300,0x012C B:01 2C	300	3	UINT16 R/W	YES
		Actual config value for AIOx:3,00 V or mA		ENTER NEW CONFIG VALUE FOR AIOx		

8.3.4.4.11 Howto detect status & diagnostic of AIOX hardware

The communication between the AIOX hardware and the main ARM co-processor is done via serial interface. Whit this command you can check if this internal connection is ok or not. If not, this highlights a severe hardware problem!

INTER PROCESSOR COMMUNICATION						
AIOX IS ONLINE	ASCII READ COMMAND	#G16AIOXISONLINE<CR> Result: #G16AIOXISONLINE:<YesNo><CR>			ASCII	
	TX	#255,G16AIOXISONLINE<CR>				
	RX	#255,G16AIOXISONLINE:YES<CR>				
		Actual communication state co-processor to AIOX processor:YES				
This command returns the actual state of the serial communication between the ARM co-processor and the additional processor for the AIOX. YES: Currently the communication is fine NO: There is a mayor problem/hardware fault between the two processors						

You can check this status also with this MODBUS register:

INTER PROCESSOR COMMUNICATION						
AIOX IS ONLINE	3x50000 4x50000 I:49999	1,0x0001 B:00 01			UINT16 R/O	
		Actual communication status co-processor to AIOX processor:OK				
This command returns the actual state of the serial communication between the ARM co-processor and the additional processor for the AIOX. =1: Currently the communication is fine =0: There is a mayor problem/hardware fault between the two processors						

Every AIOX chip controls 4 AIOX inputs or outputs. Depending on the amount of AIOX you have in your controller (4/8 or 16), you can check the SPI communication status between the AIOX processor and the chips with the ASCII commands ARECHIPSONLINE or ISCHIPONLINEx:

CHIP COMMUNICATION						
ARE CHIPS ONLINE	ASCII READ COMMAND	#ARECHIPSONLINE<CR> Result: #OK<CR>			ASCII	
	TX	#255,ARECHIPSONLINE<CR>				
	RX	#255,ARECHIPSONLINE:1,1,1,1<CR>				
		Actual state of CHIP1:1				
		Actual state of CHIP2:1				
		Actual state of CHIP3:1				
		Actual state of CHIP4:1				
This command shows the current SPI communication status with each chip. =0: Currently there is a SPI error in the communication and the chip is offline =1: The SPI communication with the chip is ok						
IS CHIPx ONLINE	ASCII READ COMMAND	#ISCHIPONLINE<CHIPNR><CR> Result: #OK<CR>			ASCII	
	CHIPNR	1				
	TX	#255,ISCHIPONLINE1<CR>				
	RX	#255,ISCHIPONLINE1:1<CR>				
		Actual state of CHIP1:1				
This command shows the current SPI communication status with chip <CHIPNR>. =0: Currently there is a SPI error in the communication and the chip is offline =1: The SPI communication with the chip is ok						

The same information for every chip can be read back with the MODBUS registers:

AIOX ONLINE						
IS ONLINE CHIP 1	3x43041 4x43041 I:43040	1,0x0001 B:00 01			UINT16 R/O	
		Is CHIPx online:1=YES				
This command shows the acutal state of the internal communication state machine for CHIPx						
IS ONLINE CHIP 2	3x43042 4x43042 I:43041	1,0x0001 B:00 01			UINT16 R/O	
		Is CHIPx online:1=YES				
IS ONLINE CHIP 3	3x43043 4x43043 I:43042	1,0x0001 B:00 01			UINT16 R/O	
		Is CHIPx online:1=YES				
IS ONLINE CHIP 4	3x43044 4x43044 I:43043	1,0x0001 B:00 01			UINT16 R/O	
		Is CHIPx online:1=YES				

Every AIOX chip has also some diagnostic information, which you can use to detect the health status of the AIOX. use GALSTATES or GLSTATEx command:

CHIP STATUS				
GET ALL LIVE STATES	ASCII READ COMMAND	#GALSTATES<CR> Result: #OK<CR>	ASCII	
	TX	#255,GALSTATES<CR>		
	RX	#255,GALSTATES:30720,28672,26624,27648,0x7800,0x7000,0x6800,0x6C00<CR>		
		Actual live state of CHIP1:30720,0x7800		
		Actual live state of CHIP2:28672,0x7000		
		Actual live state of CHIP3:26624,0x6800		
		Actual live state of CHIP4:27648,0x6C00		

Returns the actual chip status of all chips.
Each result bit stands for a different state:
Bit 0: VI_ERR_CURR_A: Status of channel A:Voltage or current error detected on Channel A. This bit is interpreted differently depending on which of the following IO function selected:
Voltage output: short-circuit error. The error condition is debounced for 2 ms before the status bit is set.
Current output: open circuit error. The error condition is debounced for 2 ms before the status bit is set.
Current input, loop powered: short-circuit error. A short to ground is detected
Current input, externally powered: short-circuit error. A current source >25 mA is detected
Bit 1: VI_ERR_CURR_B: Status of voltage input B. Same like VI_ERR_CURR_A
Bit 2: VI_ERR_CURR_C: Status of voltage input C. Same like VI_ERR_CURR_A
Bit 3: VI_ERR_CURR_D: Status of voltage input D. Same like VI_ERR_CURR_A
Bit 4: HI_TEMP_STATUS: If the die temperature is typically at or above 115°C, the HI_TEMP_STATUS bit is asserted
Bit 5: CHARGE_PUMP_STATUS: Charge pump error detected.
Bit 6: ALDO5V_STATUS: ALDO5V Power Supply Monitor Error. This bit is asserted when the ALDO5V pin falls below 4.05 V. Usually ~5V.
Bit 7: AVDD_STATUS: AVDD Power Supply Monitor Error. This bit is asserted when the AVDD pin falls below 9.26 V. Usually ~17V.
Bit 8: DVCC_STATUS: DVCC Power Supply Monitor Error. This bit is asserted when the DVCC pin falls below 1.93 V. Usually ~3.3V.
Bit 9: ALDO1V8_STATUS: ALDO1V8 Power Supply Monitor Error. This bit is asserted when the ALDO1V8 pin falls below 1.35 V. Usually ~1.8V.
Bit 10-12: ADC_CH_CURR: Current converted channel of the ADC (0:A, 1:B, 2:C, 3:D, 4:Diagnostic 0, 5:Diagnostic 1, 6:Diagnostic 2, 7:Diagnostic 3)
Bit 13: ADC_BUSY: ADC busy status bit.
Bit 14: ADC_DATA_RDY:ADC data ready. The ADC_DATA_RDY bit asserts when a conversion cycle has completed. The bit stays asserted until a user writes 1 to clear the bit. In single conversion mode, the ADC_RDY pin follows the ADC_DATA_RDY bit and only deasserts when the ADC_DATA_RDY bit is cleared. In continuous conversion mode, the ADC_RDY pin returns high after 24 µs.
Bit 15: RESERVED: Reserved

GET LIVE STATE	ASCII READ COMMAND	#GLSTATE<CHIPNR><CR> Result: #OK<CR>	ASCII	
	CHIPNR	4		
	TX	#255,GLSTATE4<CR>		
	RX	#255,GLSTATE4:26624,0x6800<CR>		
		Actual live state of CHIP4:26624,0x6800		
		Live state bit 0: VI_ERR_CURR_A:0		
		Live state bit 1: VI_ERR_CURR_B:0		
		Live state bit 2: VI_ERR_CURR_C:0		
		Live state bit 3: VI_ERR_CURR_D:0		
		Live state bit 4: HI_TEMP_STATUS:0		
		Live state bit 5: CHARGE_PUMP_STATUS:0		
		Live state bit 6: ALDO5V_STATUS:0		
		Live state bit 7: AVDD_STATUS:0		
		Live state bit 8: DVCC_STATUS:0		
		Live state bit 9: ALDO1V8_STATUS:0		
		Live state bit 10-12: ADC_CH_CURR:2		
		Live state bit 13: ADC_BUSY:1		
		Live state bit 14: ADC_DATA_RDY:0		
		Live state bit 15: RESERVED:0		

Returns the actual chip status of chip <CHIPNR>
Each result bit stands for a different state:
Bit 0: VI_ERR_CURR_A: Status of channel A:Voltage or current error detected on Channel A. This bit is interpreted differently depending on which of the following IO function selected:
Voltage output: short-circuit error. The error condition is debounced for 2 ms before the status bit is set.
Current output: open circuit error. The error condition is debounced for 2 ms before the status bit is set.
Current input, loop powered: short-circuit error. A short to ground is detected
Current input, externally powered: short-circuit error. A current source >25 mA is detected
Bit 1: VI_ERR_CURR_B: Status of voltage input B. Same like VI_ERR_CURR_A
Bit 2: VI_ERR_CURR_C: Status of voltage input C. Same like VI_ERR_CURR_A
Bit 3: VI_ERR_CURR_D: Status of voltage input D. Same like VI_ERR_CURR_A
Bit 4: HI_TEMP_STATUS: If the die temperature is typically at or above 115°C, the HI_TEMP_STATUS bit is asserted
Bit 5: CHARGE_PUMP_STATUS: Charge pump error detected.
Bit 6: ALDO5V_STATUS: ALDO5V Power Supply Monitor Error. This bit is asserted when the ALDO5V pin falls below 4.05 V. Usually ~5V.
Bit 7: AVDD_STATUS: AVDD Power Supply Monitor Error. This bit is asserted when the AVDD pin falls below 9.26 V. Usually ~17V.
Bit 8: DVCC_STATUS: DVCC Power Supply Monitor Error. This bit is asserted when the DVCC pin falls below 1.93 V. Usually ~3.3V.
Bit 9: ALDO1V8_STATUS: ALDO1V8 Power Supply Monitor Error. This bit is asserted when the ALDO1V8 pin falls below 1.35 V. Usually ~1.8V.
Bit 10-12: ADC_CH_CURR: Current converted channel of the ADC (0:A, 1:B, 2:C, 3:D, 4:Diagnostic 0, 5:Diagnostic 1, 6:Diagnostic 2, 7:Diagnostic 3)
Bit 13: ADC_BUSY: ADC busy status bit.
Bit 14: ADC_DATA_RDY:ADC data ready. The ADC_DATA_RDY bit asserts when a conversion cycle has completed. The bit stays asserted until a user writes 1 to clear the bit. In single conversion mode, the ADC_RDY pin follows the ADC_DATA_RDY bit and only deasserts when the ADC_DATA_RDY bit is cleared. In continuous conversion mode, the ADC_RDY pin returns high after 24 µs.
Bit 15: RESERVED: Reserved

The same information you can read with this MODBUS registers:

AIOX CHIP STATUS						
LIVE STATUS CHIP 1	3x43025 4x43025 I:43024	24576,0x6000 B:60 00			UINT16 R/O	
		Actual live status of CHIPx:6000				
Current live status for CHIPx. Each CHIP supports 4 AIOX channels. Each result bit stands for a different state: Bit 0: VI_ERR_CURR_A: Status of channel A:Voltage or current error detected on Channel A. This bit is interpreted differently depending on which of the following IO function selected: Voltage output: short-circuit error. The error condition is debounced for 2 ms before the status bit is set. Current output: open circuit error. The error condition is debounced for 2 ms before the status bit is set. Current input, loop powered: short-circuit error. A short to ground is detected Current input, externally powered: short-circuit error. A current source >25 mA is detected Bit 1: VI_ERR_CURR_B: Status of voltage input B. Same like VI_ERR_CURR_A Bit 2: VI_ERR_CURR_C: Status of voltage input C. Same like VI_ERR_CURR_A Bit 3: VI_ERR_CURR_D: Status of voltage input D. Same like VI_ERR_CURR_A Bit 4: HI_TEMP_STATUS: If the die temperature is typically at or above 115°C, the HI_TEMP_STATUS bit is asserted Bit 5: CHARGE_PUMP_STATUS: Charge pump error detected. Bit 6: ALDO5V_STATUS: ALDO5V Power Supply Monitor Error. This bit is asserted when the ALDO5V pin falls below 4.05 V. Usually ~5V. Bit 7: AVDD_STATUS: AVDD Power Supply Monitor Error. This bit is asserted when the AVDD pin falls below 9.26 V. Usually ~17V. Bit 8: DVCC_STATUS: DVCC Power Supply Monitor Error. This bit is asserted when the DVCC pin falls below 1.93 V. Usually ~3.3V. Bit 9: ALDO1V8_STATUS: ALDO1V8 Power Supply Monitor Error. This bit is asserted when the ALDO1V8 pin falls below 1.35 V. Usually ~1.8V. Bit 10-12: ADC_CH_CURR: Current converted channel of the ADC (0:A, 1:B, 2:C, 3:D, 4:Diagnostic 0, 5:Diagnostic 1, 6:Diagnostic 2, 7:Diagnostic 3) Bit 13: ADC_BUSY: ADC busy status bit. Bit 14: ADC_DATA_RDY:ADC data ready. The ADC_DATA_RDY bit asserts when a conversion cycle has completed. The bit stays asserted until a user writes 1 to clear the bit. In single conversion mode, the ADC_RDY pin follows the ADC_DATA_RDY bit and only deasserts when the ADC_DATA_RDY bit is cleared. In continuous conversion mode, the ADC_RDY pin returns high after 24 µs. Bit 15: RESERVED: Reserved						
LIVE STATUS CHIP 2	3x43026 4x43026 I:43025	28672,0x7000 B:70 00			UINT16 R/O	
		Actual live status of CHIPx:7000				
LIVE STATUS CHIP 3	3x43027 4x43027 I:43026	29696,0x7400 B:74 00			UINT16 R/O	
		Actual live status of CHIPx:7400				
LIVE STATUS CHIP 4	3x43028 4x43028 I:43027	25600,0x6400 B:64 00			UINT16 R/O	
		Actual live status of CHIPx:6400				

8.3.4.4.12 Howto check temperature & supply voltages of AIOX hardware

The AIOX chips deliver two supply voltage measurements and measure the internal temperature of the AIOX chips. We deliver versions with 1, 2 or 4 AIOX chips to offer 4, 8 or 16 AIOX channels. For a stable operation, this values should be under 80°C. If not, you have to coll your switchboard cabinet.

To read the current chip temperatures use this ASCII commands GCHIPTEMPS or GCHIPTEMPx. To read the average value use GAVGCHIPTEMPS or GAVGCHIPTEMPx:

GET CHIP TEMPERATURES	ASCII READ COMMAND	#GCHIPTEMPS<CR> Result: #OK<CR>	ASCII
	TX	#255,GCHIPTEMPS<CR>	
	RX	#255,GCHIPTEMPS:53.30,51.51,52.40,52.63<CR>	
		Actual temperature of CHIP1:53.30°C	
		Actual temperature of CHIP2:51.51°C	
		Actual temperature of CHIP3:52.40°C	
		Actual temperature of CHIP4:52.63°C	
This command returns for every AIOX chip the actual chip temperature in °C			
GET CHIP TEMPERATURE	ASCII READ COMMAND	#GCHIPTEMP<CHIPNR><CR> Result: #OK<CR>	ASCII
	CHIPNR	1	
	TX	#255,GCHIPTEMP1<CR>	
	RX	#255,GCHIPTEMP1:53.30<CR>	
		Actual temperature of CHIP1:53.30°C	
This command returns for AIOX chip <CHIPNR> the actual chip temperature in °C			
AVERAGE CHIP TEMPERATURES			
GET AVERAGE CHIP TEMPERATURES	ASCII READ COMMAND	#GAVGCHIPTEMPS<CR> Result: #OK<CR>	ASCII
	TX	#255,GAVGCHIPTEMPS<CR>	
	RX	#255,GAVGCHIPTEMPS:53.18,51.56,52.39,52.65<CR>	
		Average temperature of CHIP1:53.18°C	
		Average temperature of CHIP2:51.56°C	
		Average temperature of CHIP3:52.39°C	
		Average temperature of CHIP4:52.65°C	
This command returns for every AIOX chip the average chip temperature in °C			
GET AVERAGE CHIP TEMPERATURE	ASCII READ COMMAND	#GAVGCHIPTEMP<CHIPNR><CR> Result: #OK<CR>	ASCII
	CHIPNR	1	
	TX	#255,GAVGCHIPTEMP1<CR>	
	RX	#255,GAVGCHIPTEMP1:53.18<CR>	
		Average temperature of CHIP1:53.18°C	
This command returns for AIOX chip <CHIPNR> the average chip temperature in °C			

The same values can be read via MODBUS with this registers:

AIOX CHIP TEMPERATURE						
TEMPERATURE CHIP 1 IN CELSIUS	3x43001 4x43001 I:43000	443,0x01BB B:01 BB			UINT16 R/O	
		Actual measured temperature of CHIPx:44,3°C				
Current measured chip temperature for CHIPx in x*10 °C. Each CHIP supports 4 AIOX channels.						
TEMPERATURE CHIP 2 IN CELSIUS	3x43002 4x43002 I:43001	456,0x01C8 B:01 C8			UINT16 R/O	
		Actual measured temperature of CHIPx:45,6°C				
TEMPERATURE CHIP 3 IN CELSIUS	3x43003 4x43003 I:43002	440,0x01B8 B:01 B8			UINT16 R/O	
		Actual measured temperature of CHIPx:44,0°C				
TEMPERATURE CHIP 4 IN CELSIUS	3x43004 4x43004 I:43003	453,0x01C5 B:01 C5			UINT16 R/O	
		Actual measured temperature of CHIPx:45,3°C				
AIOX CHIP TEMPERATURE						
AVERAGE TEMPERATURE CHIP 1 IN CELSIUS	3x43005 4x43005 I:43004	443,0x01BB B:01 BB			UINT16 R/O	
		Measured average temperature of CHIPx:44,3°C				
Measured average chip temperature for CHIPx in x*10 °C. Each CHIP supports 4 AIOX channels.						
AVERAGE TEMPERATURE CHIP 2 IN CELSIUS	3x43006 4x43006 I:43005	456,0x01C8 B:01 C8			UINT16 R/O	
		Measured average temperature of CHIPx:45,6°C				
AVERAGE TEMPERATURE CHIP 3 IN CELSIUS	3x43007 4x43007 I:43006	439,0x01B7 B:01 B7			UINT16 R/O	
		Measured average temperature of CHIPx:43,9°C				
AVERAGE TEMPERATURE CHIP 4 IN CELSIUS	3x43008 4x43008 I:43007	452,0x01C4 B:01 C4			UINT16 R/O	
		Measured average temperature of CHIPx:45,2°C				

You can also check the AIOX chip supply voltage and the AIOX ground voltage. If the supply voltage drops under 14.5V you have a severe hardware issue or a cabling issue. If the ground voltage is not 0, you have an issue in your cabling or in the internal hardware too.

With the ASCII commands GVADDS and GVADDx you can read the current supply voltage. The average values can be read with GAVGVADDS and GAVGVADDx:

CHIP SUPPLY VOLTAGES				
GET SUPPLY VOLTAGES	**ASCII READ COMMAND**	#GVAVDDS<CR> Result: #OK<CR>	ASCII	
	TX	#255,GVAVDDS<CR>		
	RX	#255,GVAVDDS:14.66,14.66,14.66,14.65<CR>		
		Actual supply voltage of CHIP1:14.66V		
		Actual supply voltage of CHIP2:14.66V		
		Actual supply voltage of CHIP3:14.66V		
		Actual supply voltage of CHIP4:14.65V		
This command returns for every AIOX chip the actual supply voltage in Volts. This must be >14.5V, if not, there is a severe wiring or other hardware issue!				
GET SUPPLY VOLTAGE	**ASCII READ COMMAND**	#GVAVDD<CHIPNR><CR> Result: #OK<CR>	ASCII	
	CHIPNR	1		
	TX	#255,GVAVDD1<CR>		
	RX	#255,GVAVDD1:14.66<CR>		
		Actual supply voltage of CHIP1:14.66V		
This command returns for AIOX chip <CHIPNR> the actual supply voltage in Volts. This must be >14.5V, if not, there is a severe wiring or other hardware issue!				
AVERAGE CHIP SUPPLY VOLTAGES				
GET AVERAGE SUPPLY VOLTAGES	**ASCII READ COMMAND**	#GAVGVAVDDS<CR> Result: #OK<CR>	ASCII	
	TX	#255,GAVGVAVDDS<CR>		
	RX	#255,GAVGVAVDDS:14.66,14.66,14.66,14.64<CR>		
		Average supply voltage of CHIP1:14.66V		
		Average supply voltage of CHIP2:14.66V		
		Average supply voltage of CHIP3:14.66V		
		Average supply voltage of CHIP4:14.64V		
This command returns for every AIOX chip the average supply voltage in Volts. This must be >14.5V, if not, there is a severe wiring or other hardware issue!				
GET AVERAGE SUPPLY VOLTAGE	**ASCII READ COMMAND**	#GAVGVAVDD<CHIPNR><CR> Result: #OK<CR>	ASCII	
	CHIPNR	1		
	TX	#255,GAVGVAVDD1<CR>		
	RX	#255,GAVGVAVDD1:14.66<CR>		
		Average supply voltage of CHIP1:14.66V		
This command returns for AIOX chip <CHIPNR> the average supply voltage in Volts. This must be >14.5V, if not, there is a severe wiring or other hardware issue!				

Here are the MODBUS registers for the same measurements of the supply voltage:

AIOX CHIP VOLTAGES						
Vavdd CHIP 1 IN VOLT	3x43009 4x43009 I:43008	147,0x0093 B:00 93			UINT16 R/O	
		Actual measured voltage Vavdd of CHIPx:14,7V				

Current measured voltage Vavdd for CHIPx in x*10 Volts. Each CHIP supports 4 AIOX channels. This must be >14.5V, if not, there is a severe wiring or other hardware issue!						
Vavdd CHIP 2 IN VOLT	3x43010 4x43010 I:43009	147,0x0093 B:00 93			UINT16 R/O	
		Actual measured voltage Vavdd of CHIPx:14,7V				
Vavdd CHIP 3 IN VOLT	3x43011 4x43011 I:43010	147,0x0093 B:00 93			UINT16 R/O	
		Actual measured voltage Vavdd of CHIPx:14,7V				
Vavdd CHIP 4 IN VOLT	3x43012 4x43012 I:43011	147,0x0093 B:00 93			UINT16 R/O	
		Actual measured voltage Vavdd of CHIPx:14,7V				
AIOX CHIP VOLTAGES						
AVERAGE Vavdd CHIP 1 IN VOLT	3x43013 4x43013 I:43012	147,0x0093 B:00 93			UINT16 R/O	
		Measured average voltage Vavdd of CHIPx:14,7V				
Current measured voltage Vavdd for CHIPx in x*10 Volts. Each CHIP supports 4 AIOX channels. This must be >14.5V, if not, there is a severe wiring or other hardware issue!						
AVERAGE Vavdd CHIP 2 IN VOLT	3x43014 4x43014 I:43013	147,0x0093 B:00 93			UINT16 R/O	
		Measured average voltage Vavdd of CHIPx:14,7V				
AVERAGE Vavdd CHIP 3 IN VOLT	3x43015 4x43015 I:43014	147,0x0093 B:00 93			UINT16 R/O	
		Measured average voltage Vavdd of CHIPx:14,7V				
AVERAGE Vavdd CHIP 4 IN VOLT	3x43016 4x43016 I:43015	147,0x0093 B:00 93			UINT16 R/O	
		Measured average voltage Vavdd of CHIPx:14,7V				

For the ground voltage measurement you have the ASCII commands GVAGNDS and GVAGNDx. Again for the average values use GAVGVAGNDS or GAVGVAGNDx:

CHIP GROUND VOLTAGES			
GET GROUND VOLTAGES	ASCII READ COMMAND	#GVAGNDS<CR> Result: #OK<CR>	ASCII
	TX	#255,GVAGNDS<CR>	
	RX	#255,GVAGNDS:0.00,0.00,0.00,0.00<CR>	
		Actual ground voltage of CHIP1:0.00V	
		Actual ground voltage of CHIP2:0.00V	
		Actual ground voltage of CHIP3:0.00V	
		Actual ground voltage of CHIP4:0.00V	
This command returns for every AIOX chip the actual ground voltage in Volts. This must be 0, if not, there is a severe wiring or other hardware issue!			
GET GROUND VOLTAGE	ASCII READ COMMAND	#GVAGND<CHIPNR><CR> Result: #OK<CR>	ASCII
	CHIPNR	1	
	TX	#255,GVAGND1<CR>	
	RX	#255,GVAGND1:0.00<CR>	
		Actual ground voltage of CHIP1:0.00V	
This command returns for AIOX chip <CHIPNR> the actual ground voltage in Volts. This must be 0, if not, there is a severe wiring or other hardware issue!			
AVERAGE CHIP GROUND VOLTAGES			
GET AVERAGE GROUND VOLTAGES	ASCII READ COMMAND	#GAVGVAGNDS<CR> Result: #OK<CR>	ASCII
	TX	#255,GAVGVAGNDS<CR>	
	RX	#255,GAVGVAGNDS:0.00,0.00,0.00,0.00<CR>	
		Average ground voltage of CHIP1:0.00V	
		Average ground voltage of CHIP2:0.00V	
		Average ground voltage of CHIP3:0.00V	
		Average ground voltage of CHIP4:0.00V	
This command returns for every AIOX chip the average ground voltage in Volts. This must be 0, if not, there is a severe wiring or other hardware issue!			
GET AVERAGE GROUND VOLTAGE	ASCII READ COMMAND	#GAVGVAGND<CHIPNR><CR> Result: #OK<CR>	ASCII
	CHIPNR	1	
	TX	#255,GAVGVAGND1<CR>	
	RX	#255,GAVGVAGND1:0.00<CR>	
		Average ground voltage of CHIP1:0.00V	
This command returns for AIOX chip <CHIPNR> the average ground voltage in Volts. This must be 0, if not, there is a severe wiring or other hardware issue!			

Again on the MODBUS side take this registers:

AIOX CHIP VOLTAGES						
Vagnd CHIP 1 IN VOLT	3x43017 4x43017 I:43016	0,0x0000 B:00 00			UINT16 R/O	
		Actual measured voltage Vagnd of CHIPx:0,0V				
Current measured voltage Vagnd for CHIPx in x*10 Volts. Each CHIP supports 4 AIOX channels. This must be 0V, if not, there is a severe wiring or other hardware issue!						
Vagnd CHIP 2 IN VOLT	3x43018 4x43018 I:43017	0,0x0000 B:00 00			UINT16 R/O	
		Actual measured voltage Vagnd of CHIPx:0,0V				
Vagnd CHIP 3 IN VOLT	3x43019 4x43019 I:43018	0,0x0000 B:00 00			UINT16 R/O	
		Actual measured voltage Vagnd of CHIPx:0,0V				
Vagnd CHIP 4 IN VOLT	3x43020 4x43020 I:43019	0,0x0000 B:00 00			UINT16 R/O	
		Actual measured voltage Vagnd of CHIPx:0,0V				
AIOX CHIP VOLTAGES						
AVERAGE Vagnd CHIP 1 IN VOLT	3x43021 4x43021 I:43020	0,0x0000 B:00 00			UINT16 R/O	
		Measured average voltage Vagnd of CHIPx:0,0V				
Current measured voltage Vagnd for CHIPx in x*10 Volts. Each CHIP supports 4 AIOX channels. This must be 0V, if not, there is a severe wiring or other hardware issue!						
AVERAGE Vagnd CHIP 2 IN VOLT	3x43022 4x43022 I:43021	0,0x0000 B:00 00			UINT16 R/O	
		Measured average voltage Vagnd of CHIPx:0,0V				
AVERAGE Vagnd CHIP 3 IN VOLT	3x43023 4x43023 I:43022	0,0x0000 B:00 00			UINT16 R/O	
		Measured average voltage Vagnd of CHIPx:0,0V				
AVERAGE Vagnd CHIP 4 IN VOLT	3x43024 4x43024 I:43023	0,0x0000 B:00 00			UINT16 R/O	
		Measured average voltage Vagnd of CHIPx:0,0V				