

Communication in MODBUS TCP network (Gen1)

Abstract

The Application note describes the use of MODBUS TCP protocol in Generation 1 control systems using a table definition. The Application note deals with both communication with other products in Slave mode as well as with communication with a superior Master station.

Author: Michal Kupčik
Document: ap0059_ap_en_002.pdf

Attachments

File contents: ap0059_ap_en_002.zip

modbstcp_p1_cz_100.dso	Example of a control system parametrization as master.
modbstcp_p2_cz_101.dso	Example of master communication with six slave stations without TCP connection failure treatment.
modbstcp_p3_cz_100.dso	Example of a control system parametrization as slave.
modbstcp_p4_cz_100.dso	Example of master communication with six slave stations including treatment of not established TCP connection.

Contents

	Contents.....	2
	Revision history.....	3
	Related documentation.....	3
	Purpose of the application note.....	3
1	Definitions of terms	4
2	MODBUS protocol.....	5
2.1	Supported MODBUS functions	5
3	Connecting the communication network.....	6
4	Time conditions in the network.....	8
4.1	Communication period	8
	Example	8
4.1.1	Communication priorities.....	9
	Automatic reading priority	9
	Automatic writing priority	9
	Manual communication priority	9
4.1.2	Gathering communication frames	10
	Example	10
4.2	Communication in the event of a connection failure	10
5	Control system as Master	11
5.1	Communication definition.....	11
5.2	Definition of a Slave station and data points for communication.....	12
5.3	Automatic communication.....	13
5.4	Manual communication	14
5.5	Communication statuses.....	15
5.6	Example of a control system parametrization as Master	16
5.7	Master communication with more than 3 Slave stations	18
5.8	Master communication with more than 3 Slave stations including TCP connection failure treatment	20
6	Control system as Slave.....	22
6.1	Communication definition.....	22
6.2	Definition of data points for communication	23
6.3	Communication statuses.....	23
6.4	Example of a control system parametrization as Slave	23
7	Appendix A	26
7.1	Compatibility with communication initialization via modules	26
7.1.1	MODBUS Master	26
7.1.2	MODBUS Slave	26
8	Technical support	27
9	Warning.....	28

Revision history

Version	Date	Changes by	Changes
001	25. 03. 2019	Kupčík M.	New document.
002	22. 03. 2022	Kupčík M.	Text edited, new chapter 5.8.

Related documentation

1. Help tab in the PseDet section of the DetStudio development environment
file: PseDet_en.chm
2. Application note AP0037 – Principles of Ethernet network usage
file: ap0037_en_xx.pdf

Other documentation available at the time of publication of this document:

3. RFC 793 Transmission Control Protocol – Protocol specification
file: <https://www.ietf.org/rfc/rfc793.txt>

Purpose of the application note

The application note is intended for Generation 1 control systems – control systems with the NOS operating system programmed in DetStudio in the PseDet editor. More information about the system generations can be found on the amitautomation.cz website.

1 Definitions of terms

Control system

These are Generation 1 control systems and terminals from AMiT, where process algorithms are programmed in the so-called PseDet part of the DetStudio environment. E.g. **AMiNi4DW2**, **AD-CPUW2**, **AMAP99W3** or **ART4000W3**.

Master station

This station actively communicates with Slave stations. In TCP/IP terminology, it is designated as Client.

Slave station

It is a station with a unique address which passively listens on the communication interface and responds only after receiving a particular frame from the Master. In TCP/IP terminology, it is designated as Server.

Data point

It is a definition of a register (input or holding) or binary (input or output) which usually represents an input or output on a Slave station. Each data point is directly assigned a (matrix) variable or bit into which read values are to be written or from which values for writing into the Slave station shall be taken.

2 MODBUS protocol

MODBUS is an open communication protocol developed by the Modicon company. Originally, the protocol was designed for an RS232 bus; however, it soon transitioned to RS485 because of its better reliability and options of connecting multiple devices at longer distances. Throughout its development, MODBUS has been extended with description of Ethernet interface implementation. Port 502 has been designated as a standard TCP port. The protocol is flexible but at the same time easy to implement, and therefore soon various producers started implementing it into their devices. Today, not only microcontrollers or industrial PCs, but also many intelligent sensors, actuators and other simple components enable MODBUS communication.

AMiT supports MODBUS TCP communication in control systems with the letter “W” in the name. The master/slave determination depends on a specific implementation.

Note

When defining the MODBUS TCP communication protocol, other communication protocols can be used on the Ethernet interface.

Caution!

Various producers may have various interpretations of data point addressing, despite MODBUS protocol specification. Find out more in the Help section of DetStudio called “PseDet – Creating control processes”, in chapter “Contents/Communication/Modbus” in the section “Addressing Registers/Binaries”.

Caution!

To support MODBUS TCP communication, the NOS operating system of at least version 3.70 must be loaded in the control system.

2.1 Supported MODBUS functions

The following MODBUS protocol functions are supported in AMiT control systems. Functions stem from the MODBUS protocol definition and define the type of the frame used.

Function No.	Description
1	Read one/multiple binary outputs.
2	Read one/multiple binary inputs.
3	Read one/multiple holding registers.
4	Read one/multiple input registers.
5	Write one binary output.
6	Write one holding register.
15	Write multiple binary outputs.
16	Write multiple holding registers.

The stated description is only general and for orientation. Specific descriptions of individual functions depend on specific type of device.

Very often pairs of registers are used for analogue values, so the writing of analogue outputs is done using function 16 Find out more in the Help section of DetStudio called “PseDet – Creating control processes”, in chapter “Contents/Communication/Modbus” in the section “Communication Points Mapping”.

3 Connecting the communication network

In order for the entire MODBUS network to work properly, it is necessary to design, connect and configure individual network modules and to programme communication in control systems.

When wiring the network on Ethernet interface, it is necessary to follow the recommendations stated in Application note AP0037 – Principles of Ethernet network usage.

An AMiT control system may behave as Master or Slave in a MODBUS network. In the Master role, typically in combination with technological devices from other manufacturers (e.g. actuators) or in the Slave role as part of larger networks.

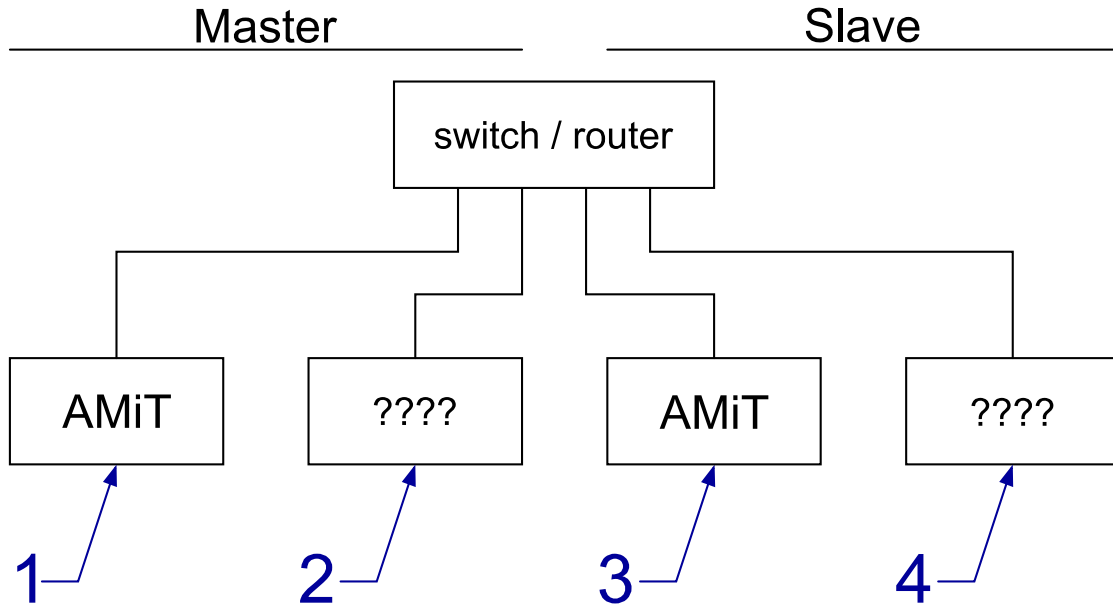


Fig. 1 – Communication via protocol MODBUS TCP via network element

Legend

Number	Significance
1	AMiT control system as a Master station
2	Third-party devices as Master stations
3	AMiT control system as a Slave station
4	Third-party devices as Slave stations

The converter MODBUS TCP / MODBUS RTU may be used as a Slave station.

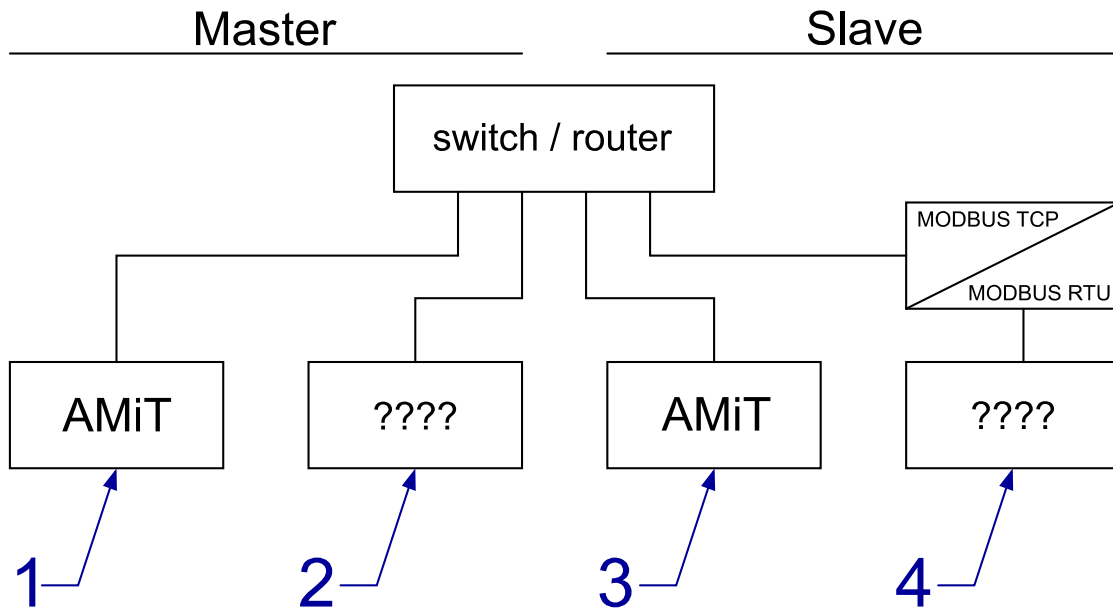


Fig. 2 – Communication via protocol MODBUS TCP with a converter from TCP to RTU

Legend

Number	Significance
1	AMiT control system as a Master station
2	Third-party devices as Master stations
3	AMiT control system as a Slave station
4	MODBUS RTU device as a Slave station

4 Time conditions in the network

Communication is executed within certain periods. At the beginning of such period, master activates the interface (based on the definition table, a package of requests is created), and subsequently requests for each remote point stated in the definition table are communicated. The interface activity finishes when the last frame is communicated.

4.1 Communication period

The period of communication via Ethernet interface generally strongly depends on other traffic in the Ethernet network. Under high traffic load, some packets may even get lost. The TCP protocol includes implemented mechanisms to detect such potential losses and re-send the lost packets.

While in the RTU specification of the MODBUS protocol the time ratios are clearly given, this does not apply in the case of TCP communication. Furthermore, they also depend on the method of implementation of the so-called TCP stack on part of both Master and Slave station. For these reasons, communication periods are generally undeterminable.

The following table shows recommended times for communication period calculations. However, these apply to communication between AMiT-made control systems and with medium network load.

Minimum communication period [ms]	Data point communication period [ms]
5	0.2 × register, 0.1 × each set of eight binaries started

Example

Need to communicate periodically with three Slave stations. Each requires 8 registers and 18 bins to communicate. Thus, two lines of remote point communication (one group of registers or binaries) will be defined for each station. The calculation will be based on table values. This implies the assumption of a similar TCP stack implementation and low network load.

$$T_{reg} = 5 + 0.2 \times 8 = 6.6 \text{ ms}$$

$$T_{bin} = 5 + 0.1 \times 3 = 5.3 \text{ ms}$$

In the case of the T_{bin} calculation, it is based on the fact that 18 binaries are divided into 8 + 8 + 2, i.e. 3 is taken as the value of each octet.

The total minimum communication period is therefore:

$$T_{total} = 6.6 \times 3 + 5.3 \times 3 = 35.7 \text{ ms}$$

Amendment

Completely identical times would result if only one Slave station was communicated with, and for this Slave station, the given layout of 3 × 8 registers and 3 × 18 bins was communicated on six rows of the definition table.

Warning!

It is recommended to apply the aforementioned calculations with max. 3 Slave stations, or more specifically for threesomes of Slave stations at maximum. More about this is described in the chapters 5.7 “Master communication with more than 3 Slave stations” and 5.8 “Master communication with more than 3 Slave stations including TCP connection failure treatment”.

4.1.1 Communication priorities

Automatic reading priority

DetStudio offers three reading priorities for automatic reading of values from Slave stations:

Reading priority	Communication period [ms]
Low	5,000
Normal	1,000
High	200

By defining the priority, the programmer selects with what period the given row of the table is to be read.

It applies that the NOS operating system goes through individual definition tables every 200 ms and if it discovers a row with automatic reading priority and is to communicate this row in the given 200 ms cycle, the given row is placed in a communication request queue.

Automatic writing priority

In case of automatic writing, there is no defined priority with a time period; the only available option is switching to priority **Auto**.

If this priority is selected, the assigned variable (even with the same value) is marked upon each writing and placed at the start of the communication request queue. Writing requests are therefore always communicated before reading requests.

Due to internal mechanisms for detection of writing into the assigned variable, the given variable may be used in the definition table with priority **Auto** only once. If, for example, it is required that values from multiple cells of a matrix variable are used to write different registers or values of bits of an integer variable are used to write different binaries, the definition can be solved based on the register layout:

- ◆ If two writing registers or binaries are in a sequence one after another, define only one definition row and have a set value of the column **Number** to the corresponding value. An example of such a definition is available in the Help section of DetStudio called "PseDet – Creating control processes", in chapter "Contents/Communication/Modbus – Device table editor" in the section "Notes".
- ◆ If writing registers or binaries are not in a sequence, it is necessary to set communication priorities for the given definition rows manually. In order to detect a binary value change, you can use the module **BinDiff**.

Manual communication priority

If automatic communication priority of definition rows does not permit a correct requested mode of communication with the Slave station, it is necessary to use manual communication priority. This is set by the `--manual--` option in the desired communication priority column.

The following modules are used to launch manual communication:

- ◆ **MdbmMark** – marking a rather large number of definition rows for communication,
- ◆ **MdbmRead** – marking a specific definition row for reading,
- ◆ **MdbmWrite** – marking a specific definition row for writing,
- ◆ **MdbmWrBeg**, **MdbmWrFin** – marking a specific definition row for a so-called safe writing.

Modules work with labels on a specific Device definition as well as with labels for a specific definition row, except for the module **MdbmMark**. More information on individual modules is available in the Help section of DetStudio called "PseDet – Creating control processes", in descriptions of individual modules.

Unlike in case of automatic communication priority, data points in manual communication are communicated immediately after execution of the given module. Using these modules it is therefore possible to achieve communication with a faster period than 200 ms.

4.1.2 Gathering communication frames

When calculating a minimum communication period, it is also necessary to consider automatic gathering of communication frames going in a sequence when using communication functions 1, 2, 3, 4, 15 and 16 (see chapter 2.1 “Supported MODBUS functions”).

Example

Let's have a definition table with two rows for reading two registers into two variables. If addresses of the given registers are not in a sequence, e.g. addresses 0 and 2, communication is executed in two frames. Using the table values from the beginning of this chapter, calculate the time:

$$T = 2 \times (5 + 0.2 \times 1) = 10.4 \text{ ms}$$

However, if register addresses are defined in a sequence, e.g. addresses 0 and 1, communication is executed in a single frame. Using the table values from the beginning of the chapter, calculate the time:

$$T = 5 + 0.2 \times 2 = 5.4 \text{ ms}$$

If we go back to “**Amendment**” of the original example at the beginning of this chapter (**Example**), then in case all 3×8 registers and 3×18 binaries were defined in an uninterrupted sequence, e.g. in registers with addresses 0 to 7, 8 to 15 and 16 to 23 then registers and binaries would be communicated each in a single frame. Using the table values from the beginning of the chapter, calculate the time:

$$T = (5 + 0.2 \times 18) + (5 + 0.1 \times 9) = 14.5 \text{ ms}$$

4.2 Communication in the event of a connection failure

If communication with the Slave station is not available, or more specifically if there has been no response to the request, the following algorithm of communication with this station is launched:

1. The frame that received no response is repeated $2 \times$ more.
2. If there is still no response, subsequent communication requests are ignored for the period of 15 seconds. Ignorance of requests is signalled by setting bit No. 4 of the parameter “Status” of the module `MdbmReqSt` (for description, see chapter 5.5 “Communication statuses”) to True.
3. After the communication request ignorance period elapses, the table of communication requests of the given Slave station is checked. If any request is found, attempt to communicate it is made. The first items to be checked are writing requests. At the same time, bit No. 4 of the parameter “Status” of the module `MdbmReqSt` is set to False for the communication period.
4. If there is still no correct response, the current time of ignoring communication requests is prolonged by 2 seconds. The bit No. 4 of the parameter “Status” of the module `MdbmReqSt` is set to True again.
5. If it was a writing communication request, the request maintains its flag for communication after the delay time has elapsed. If it was a reading request, the flag for communication is cancelled.
6. After a new delay time elapses, the algorithm repeats from the point 3. The maximum time to ignore communication requests is 30 s. Hence a series of times of 15 s, 17 s, 19 s, ..., 29 s, 30 s,

5 Control system as Master

The definition of communication via MODBUS protocol in the role of Master is done by means of three definitions:

- ◆ creating a communication definition of the protocol in the Master role,
- ◆ creating a definition of the Slave station,
- ◆ defining data points of the given Slave station for communication.

5.1 Communication definition

Creating a communication definition of MODBUS protocol in the Master role represents inserting a definition of the communication item **ModbusMaster** into the application. The insertion is done in DetStudio in the Project window in the “Project/Communication/Modbus” node. When the context menu above this item is called, select the **Add Master** item.

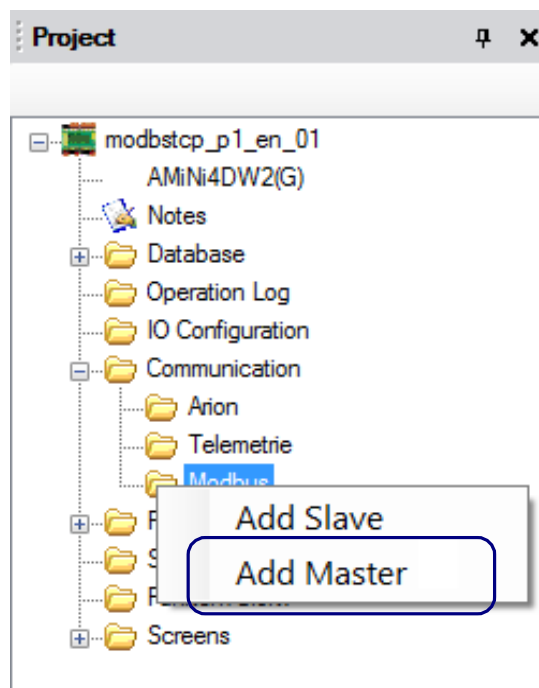


Fig. 3 – Item “Add Master” in the definition of “Modbus” communication

After the Master definition is added, a communication node **ModbusMaster0** is created with the following properties values:

- ◆ **BaudRate:** 19,200
- ◆ **Mode:** SerialLineRTU
- ◆ **Parity:** Even
- ◆ **SerialPort:** 0
- ◆ **StopBit:** One
- ◆ **ToReceive:** 30
- ◆ **ToTransmit:** 4

In order to communicate via MODBUS TCP protocol, it is necessary to set the value of the property **Mode** to “Modbus_TCP”.

More information is available in the Help section of DetStudio called “PseDet – Creating control processes”, in chapter “Contents/Communication/Modbus/Master – creating and setting general parameters”.

5.2 Definition of a Slave station and data points for communication

For communication with individual Slave stations we need to define its address in the MODBUS network and a list of communication points.

Individual Slave stations called *Device* are defined in the Project window directly in the node of the specific **ModbusMasterX** definition. After calling the context menu above the selected item, select the item **Add Device**.

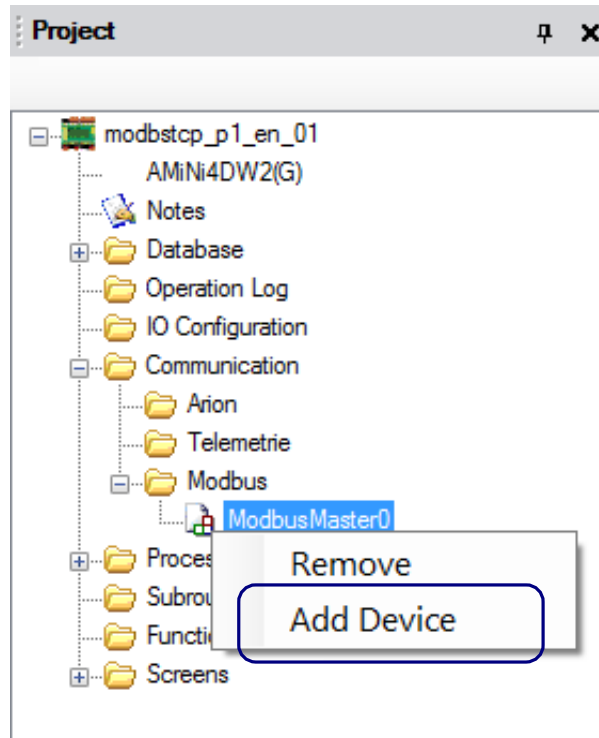


Fig. 4 – Item “Add Device” in the definition of the node “ModbusMasterX”

After the definition is added, a communication node **ModbusDevice0** is created with default values of properties:

- ◆ **Address:** 1
- ◆ **ByteOrder:** 0-1-2-3 (Modbus default)
- ◆ **ClientLabel:** -1

Value of the property **Address** usually has no significance in definitions of Slave stations that do not serve as converters from MODBUS TCP to MODBUS RTU.

Since 32-bit types (Long and Float) are not defined in the MODBUS protocol in any way, the manner of these extension register implementation (if any) is only up to the given device’s manufacturer. Due to the fact that the sequence of bytes in 16-bit words is defined as Big-Endian, in AMiT products this manner of coding has been also applied to the aforementioned 32-bit types.

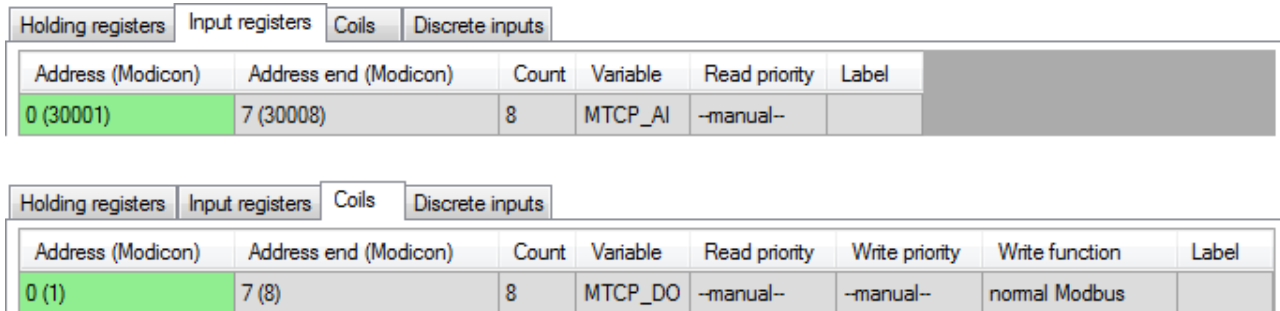
If communication of 32-bit values results in values in variables significantly different than actual values in the Slave station, it is recommended to change the value of the property **ByteOrder**, usually to 2-3-0-1.

Value of the property **ClientLabel** is used in case of manual communication with the Slave station and when determining communication statuses (see chapters 5.4 “Manual communication” and 5.5 “Communication statuses”).

After you define the node **ModbusDeviceX**, you can double-click it in the Project window and call up the definition table of communication data points of this Slave station.

In the following three chapters we will consider communication with a Slave station, where the requirement is to read analogue values and write binary values. The operation manual for this module states that analogue values are read using function 4 – reading input registers; writing into digital outputs is done through function 5 – setting one binary output (coil) or 15 – setting binary outputs (coils). The aforementioned functions show that the definition table of the node **ModbusDeviceX** includes tabs “Input registers” and “Coils”.

The definition of individual data points can be done e.g. by dragging and dropping from the ToolBox window. More information on definition of data points is available in the Help section of DetStudio called “PseDet – Creating control processes”, in chapter “Contents/Communication/Modbus – Device table editor”.



The figure shows two screenshots of the Modbus device table editor. The top screenshot shows the 'Input registers' tab with a table where the 'Read priority' column is set to '-manual-'. The bottom screenshot shows the 'Coils' tab with a table where the 'Write priority' column is set to '-manual-' and the 'Write function' is 'normal Modbus'.

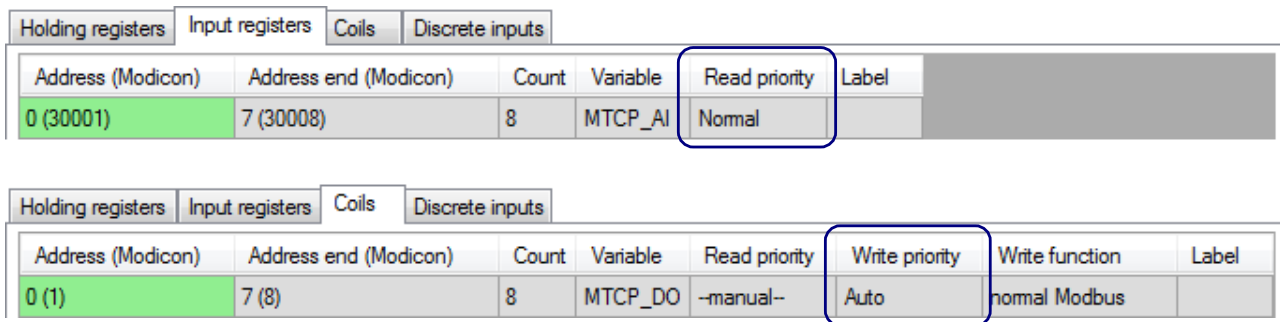
Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Label
0 (30001)	7 (30008)	8	MTCP_AI	-manual-	

Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Write priority	Write function	Label
0 (1)	7 (8)	8	MTCP_DO	-manual-	-manual-	normal Modbus	

Fig. 5 – Basic definition of data points and assigned variables

5.3 Automatic communication

After defining data points, items --manual-- are pre-set in the definition table columns “Reading priority” and “Writing priority”. For automatic communication, it is necessary to change their settings to one of the automatic priorities stated in chapter 4.1.1 “Communication priorities”.



The figure shows two screenshots of the Modbus device table editor with specific settings highlighted by blue boxes. In the top screenshot, the 'Read priority' for the input register is set to 'Normal'. In the bottom screenshot, the 'Write priority' for the coil is set to 'Auto'.

Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Label
0 (30001)	7 (30008)	8	MTCP_AI	Normal	

Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Write priority	Write function	Label
0 (1)	7 (8)	8	MTCP_DO	-manual-	Auto	normal Modbus	

Fig. 6 – Definition of data points for automatic communication

Recommendation

In the case when it is not desirable to write even if the same value is written to the variable, it is possible to recommend the following code that writes to the communication variable only when the value of the auxiliary working variable changes:

```

If MTCP_DO != MTCP_DO_pr
  Let MTCP_DO_pr = MTCP_DO
EndIf

```

In application, this auxiliary working variable is to be used in the application code, whereas the communication variable will only be used in the definition table.

Note

Due to internal algorithms for the use of automatic writing priority, it is not suitable to have both automatic reading priority and automatic writing priority on a single row. More information is available in the Help section of DetStudio called "PseDet – Creating control processes", in chapter "Contents/Communication/Modbus – Device table editor" in the section "Notes".

5.4 Manual communication

For manual communication, it is necessary to define labels. There are two types of labels:

- ◆ Slave station definition label – property `ClientLabel`,
- ◆ definition row label – column "Label".

Value label values must not be negative. The label `ClientLabel` must be unique within the application, the label in the data points definition table must be unique within the given table.

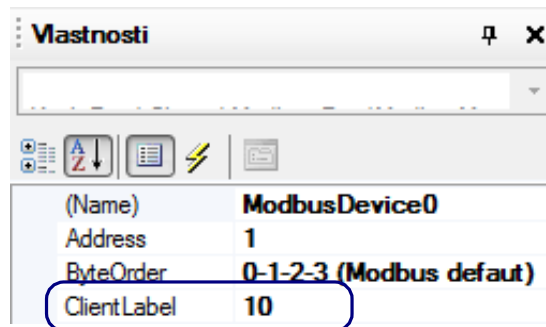


Fig. 7 – Label `ClientLabel` definition

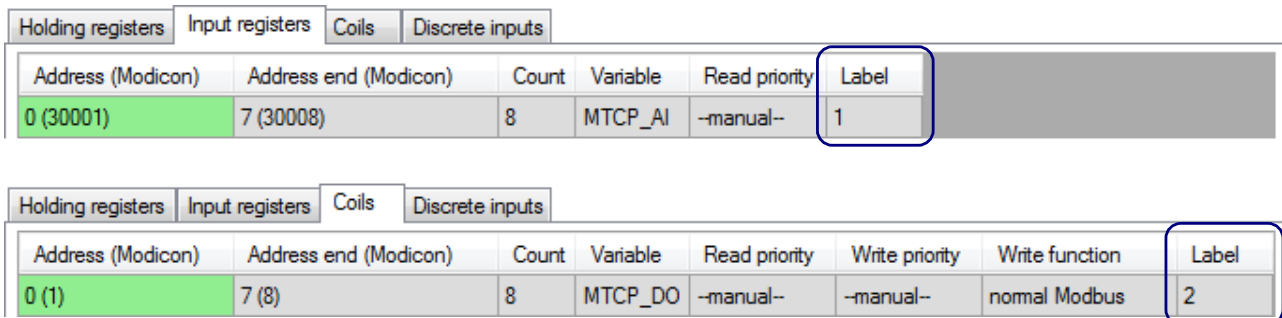


Fig. 8 – Data points label definition

As soon as the labels are defined, it is possible to use modules `Mdbm***` mentioned in chapter 4.1.1 "Communication priorities", section "Manual communication priority".

The following definition can be used to indicate a line with label 1 for reading analogue inputs:

```
MdbmRead 10, 1, MTCP_AI_rslt
    |
    |   L Module execution result
    |   L Definition row label
    |   L ClientLabel
```

or:

```
MdbmMark 1, 4, 0, 8, MTCP_AI_rslt
    |
    |   |
    |   |   L Module execution result
    |   |   L Number of addresses for marking
    |   |   L Starting address for marking
    |   |   L Communication function definition
    |   |   L ClientLabel
```

Using the module **MdbmMark** obviously does not require a definition for labels for specific rows. The module therefore allows us to batch-label a large number of definition rows of a single group of data points.

The following definition can be used to indicate a line with label 2 for reading digital inputs:

```
MdbmWrite 10, 2, MTCP_DO_rslt
    |
    | | L Module execution result
    | | L Definition row label
    | L ClientLabel
```

or:

```
MdbmMark 1, 5, 0, 8, MTCP_DO_rslt
```

In this example, it is not necessary to consider the necessity to use a so-called safe writing by means of modules **MdbmWrBeg** and **MdbmWrFin**, because the given definition row serves only for writing.

Note

It is possible to have both automatic and manual communication defined on one definition row in the table. These two communications are not mutually exclusive.

5.5 Communication statuses

We use the following modules to detect communication statuses:

- ◆ **MdbmCliSt** – detecting the communication interface status; a communication interface is presumed as the combination IP address + TCP port,
- ◆ **MdbmReqSt** – detecting the communication status of a specific communication request.

Module **MdbmReqSt** can be used for detection of failed communication with the Slave station using bit No. 4 (see the text under tables). In order to be able to detect a communication failure, the label of the most frequently communicated definition row of the table is used.

```
MdbmCliSt 10, MTCP_ClSt, MTCP_CS_rslt
    |
    | | L Module execution result
    | | L Status of the client, or more specifically of the communication
interface
    | L ClientLabel
```

```
MdbmReqSt 10, 1, MTCP_RqSt, MTCP_RS_rslt
    |
    | | L Module execution result
    | | L Communication request status
    | | L Definition row status
    | L ClientLabel
```

Using the module **MdbmReqSt** can be clearly recommended when debugging communication, when the value of the communication request state can be used to obtain information about a possible communication error.

The value of the communication request status gets various bit-coded values depending on the current status of the data point communication request entry and on the current communication status according to the following table.

Bit	Significance
0	Has value 1 if communication is currently in progress.
1	Has value 1 if the previous finished communication has finished successfully.
2	Has value 1 if the previous finished communication has finished in an error.
4	Has value 1 if the next attempt for communication is ignored.
6	Has value 1 if the request is marked for reading and awaits communication.
7	Has value 1 if the request is marked for writing and awaits communication.
8	Has value 1 if the request has been blocked by the module MdbmWrBeg .
9	Has value 1 if the request is repeatedly automatically marked for writing and awaits communication.
10	Has value 1 if the currently communicated request is for writing.
11	Has value 1 if the previous finished communication was for writing.
12 to 15	If the communication ended up in an error (bit 2 has value 1), these bits contain the communication error codes according to the following table. Otherwise, values of these bits are not defined.

Error codes in bits 12 to 15

Error code	Significance
0	Station responded negatively, with an unspecified error.
1	Station response: "Incorrect function".
2	Station response: "Incorrect register/binary address".
3	Station response: "Incorrect data value."
4	Unknown unspecified error.
5	Station has not responded within the required period.
6	Transmission error (incorrect CRC, incorrect response length, etc.).
7	Connection error, usually in case of MODBUS TCP communication.

With respect to the fact that bit No. 4 only signals ignoring of a subsequent communication, using the module **RS** to signalize a failure in communication with the given Slave station is recommended

`RS MTCP_RS_rslt.4, MTCP_RS_rslt.1, MTCP_Problem.0`

5.6 Example of a control system parametrization as Master

Let's have an application in which one control system **AMiNi4DW2** is to serve as a Slave station for the second control system **AMiNi4DW2**. Slave station's register layout is described in chapter 6.4 "Example of a control system parametrization as Slave".

The distribution of holding registers is known:

- ◆ address 0 – DI,
- ◆ addresses 1 to 8 – AI_Integer,
- ◆ addresses 10 to 25 – AI_Float,
- ◆ address 100 – DO,
- ◆ addresses 101 to 104 – AO.

First, we create the variables that will be assigned to the given definition rows:

- ◆ **AMiNi_DI** – type I,
- ◆ **AMiNi_AI** – type MI, dimension [1×8],
- ◆ **AMiNi_AI_F** – type MF, dimension [1×8],
- ◆ **AMiNi_DO** – type I,
- ◆ **AMiNi_AO** – type MI, dimension [1×4].

The next step is to create a definition node **ModbusMaster0** and within it create a definition **ModbusDevice0**. Let the IP address of the control system be 192.168.1.2, communication point is default 502.

In the definition of the `ModbusDevice0`, table we define 5 rows on the “Holding registers” page. In the definition lines, select the priorities as follows:

- ◆ address 0 – automatic reading with priority **Normal**,
- ◆ addresses 1 to 8 – automatic reading with priority **Low**,
- ◆ addresses 10 to 25 – automatic reading with priority **Low**,
- ◆ address 100 – manual writing,
- ◆ addresses 101 to 104 – automatic writing.

Because manual writing is defined and the request for detection of connection status with the Slave station, go to the definition of `ModbusDevice0` and define a parameter `ClientLabel1`, in it, e.g. to value 10. For detecting the connection status of the Slave station we define a label in the register line with address 0 and for manual start of communication we define a label in the register line with address 100.

Due to the fact that AMiT product support communication frames 6 as well as 16, the column “Writing function” maintains the option **normal Modbus**.

The resulting table’s appearance is illustrated in the following image.

Holding registers								
Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Write priority	Write function	Label	
0 (40001)	0 (40001)	1	AMiNi_DI	Normal	-manual-	normal Modbus	1	
1 (40002)	8 (40009)	8	AMiNi_AI	Low	-manual-	normal Modbus		
10 (40011)	25 (40026)	16	AMiNi_AI_F	Low	-manual-	normal Modbus		
100 (40101)	100 (40101)	1	AMiNi_DO	-manual-	-manual-	normal Modbus	2	
101 (40102)	104 (40105)	4	AMiNi_AO	-manual-	Auto	normal Modbus		

Fig. 9 – Basic definition of data points and assigned variables

The last step is defining modules `MdbmReqSt` and `MdbmWrite`. In order to prevent excessive communications of register 100, use the algorithm described in chapter 5.4 “Manual communication”.

The resulting code in the periodic process is written as follows:

```
MdbmReqSt 10, 1, AMiNi_RqSt, AMiNi_RS_rs

If not AMiNi_RqSt.4
  If AMiNi_DO != AMiNi_DO_pr
    Let AMiNi_DO_pr = AMiNi_DO
    MdbmWrite 10, 2, AMiNi_DO_rs
  EndIf
EndIf
```

Note

The application code should subsequently deal with conversion of values of analogue quantities, which is not considered in this example.

The algorithm is included in the appendix `ap0059_ap_en_xxx.zip`. This is a sample application called “`modbstcp_p1_en_xxx.dso`” created in the DetStudio development environment. This application has been created for the control system **AMiNi4DW2**. However, it can be modified to suit any control system with the letter “W” included in the system name using the DetStudio menu “Tools/Change station”.

5.7 Master communication with more than 3 Slave stations

In the case of communication with more than three Slave stations, the limitations of the implemented TCP stack cause switching between the defined IP addresses.

Under such circumstances, communication failure may occur from time to time in some definition rows according to the selected priority.

If behaviour described above keeps occurring repeatedly, it is recommended to prevent this random behaviour by dividing individual Slave stations into groups of one to three stations and to communicate the given groups with manual communication priority.

The **Switch** module in combination with the auxiliary counting variable can be used for this purpose.

As an example, we will use the communication with six **AMiNi4DW2** control systems with the application described in chapter 6.4 “Example of a control system parametrization as Slave”. The base application is the one created in the previous chapter.

In order to avoid the need to define for each Slave station its own complete set of variables, the property of the assigned matrix variable is used, where the loaded registers are written column by column and only after writing to the last column is continued on the next row. Modify the loading variables to:

- ◆ **AMiNi_DI** – type MI, dimension [6×1],
- ◆ **AMiNi_AI** – type MI, dimension [6×8],
- ◆ **AMiNi_AI_F** – type MF, dimension [6×8].

For writing DO, comparison with the previous value is used and therefore the corresponding variables can be converted to matrix variables:

- ◆ **AMiNi_DO** – type MI, dimension [6×1],
- ◆ **AMiNi_DO_pr** – type MI, dimension [6×1].

For writing AO, it would also be possible to convert to a matrix of six rows. However, at the same time conversion to a single matrix variable would necessitate manual periodic writing into all Slave stations. In order to maintain the functionality of the original **Auto Write Priority**, the **VarWStat** module will be used to detect the writing to the variable and only on the basis of this information will the writing to the corresponding Slave station occur. In order to function correctly, each Slave station must have its own **AMiNiX_AO** variable.

After modifying the variables, it is necessary to modify the assigned variables in the Slave station definition, or the cells defining the working row of the variable and set the read and write priorities to **--manual--**.

Holding registers		Input registers	Coils	Discrete inputs				
Address (Modicon)	Address end (Modicon)	Count	Variable	Read priority	Write priority	Write function	Label	
0 (40001)	0 (40001)	1	AMiNi_DI[0,0]	--manual--	--manual--	normal Modbus	1	
1 (40002)	8 (40009)	8	AMiNi_AI[0,0]	--manual--	--manual--	normal Modbus		
10 (40011)	25 (40026)	16	AMiNi_AI_F[0,0]	--manual--	--manual--	normal Modbus		
100 (40101)	100 (40101)	1	AMiNi_DO[0,0]	--manual--	--manual--	normal Modbus		
101 (40102)	104 (40105)	4	AMiNi1_AO	--manual--	--manual--	normal Modbus		

Fig. 10 – Adjusted definition of assigned variables

After modifying the Slave station definition according to this pattern, we create five more definitions. In the five **ModbusMasterX** nodes we create one **ModbusDeviceX** in each node. In all of them, **ClientLabel** priorities must be defined with unique values.

The next step is to create an algorithm for sequential communication with individual Slave stations.

First, it is recommended to calculate maximum communication period in the worst communication scenario of all data points and without merging communication frames:

$$T = (5 + 0.2 \times 1) + (5 + 0.2 \times 8) + (5 + 0.2 \times 16) + (5 + 0.2 \times 1) + (5 + 0.2 \times 4) = 31 \text{ ms}$$

Next, select the Slave station groups for communication. In this example, we choose that there is only one station per group. The previous calculation shows that the given algorithm for switching communication of Slave stations may be in the periodic process with a period at least 31 ms. We choose the algorithm to be in a regular periodic process. Common periodic processes have the shortest period possible 100 ms. However, this lowest time cannot be used as the process period, since it takes some time to establish a TCP connection. The minimum recommended time is 3,000 ms.

The next step is to write code for communication with the station. The code may look as follows:

```
// Periodic reading
MdbmMark 10, 3, 0, 26, NONE
// Event writing
// DO
If AMiNi_DO[0,0] != AMiNi_DO_pr[0,0]
    Let AMiNi_DO_pr[0,0] = AMiNi_DO[0,0]
    MdbmMark 10, 6, 100, 1, NONE
EndIf
// AO
VarWStat AMiNi1_AO, @AO_w, 0
If @AO_w
    MdbmMark 10, 6, 101, 4, NONE
EndIf
```

The last step is copying the code into six **Case** branches of the module **Switch**, and adjusting the code so as to link **Client** parameters, matrix cells and assigned variables correctly and provide gradual switching of the branches.

```
Switch SlaveNdx
    Case 0
        // Periodic reading
        MdbmMark 10, 3, 0, 26, NONE
        ...
    EndCase

    Case 1
        // Periodic reading
        MdbmMark 20, 3, 0, 26, NONE
        ...
    EndCase

    Case 2
        ...
    EndCase

    ...

    Case 5
        ...
    EndCase

EndSwitch

Let SlaveNdx = If(SlaveNdx < 5, SlaveNdx + 1,0)
```

The algorithm is included in the appendix ap0059_ap_en_xxx.zip. This is a sample application called “modbstcp_p2_en_xxx.dso” created in the DetStudio development environment. This application has been created for the control system **AMiNi4DW2**. However, it can be modified to suit any control system with the letter “W” included in the system name using the DetStudio menu “Tools/Change station”.

5.8 Master communication with more than 3 Slave stations including TCP connection failure treatment

However, the simple algorithm presented in the previous section can only be used if it is guaranteed that the Slave stations are always available for TCP communication. In the case when communication breakdowns may occur, e.g. due to power failures or overload of the Ethernet network, depending on the length of the outage, communication with other connected Slave stations may also be broken by the control system.

To avoid this, it must be ensured that after the TCP connection with one Slave station is broken, a subsequent attempt to communicate with this station is made after a longer period of time. The TCP protocol specification defines an “MSL” time of 2 minutes. However, for a complete transition from the TCP state “TIME-WAIT” it is necessary to wait at least twice this time, the so-called “2MSL” time. In order to ensure complete termination of communication on this TCP connection, an extra minute is added to the wait. After a communication breakdown, the next attempt to communicate with this Slave station should be made in 5 minutes at the earliest.

Under standard circumstances, there is capacity reserved in the control system for three TCP connections. From this it can be deduced that in case the communication with one Slave station breaks down, one of these TCP connections will be occupied for at least the above mentioned time “2MSL”. Thus, the communication breakdown with one Slave station is not critical for the control system. In the event that the second Slave station also fails, this may indicate more serious problems on the network and in such a situation it is recommended to perform a complete timeout on all TCP connections for the recommended 5 minutes.

Because the TCP handshake can take some time, it is necessary to wait some time after sending the communication request for the actual communication. The time ratios mentioned in the chapter 4.1 “Communication period” apply to already established TCP connections. With a margin of 10 seconds can be recommended as the maximum waiting time between activating a communication request to a new Slave station and evaluating whether the TCP connection has been successfully established.

Since the write request results in repeated communication attempts in case of unsuccessful communication (see chapter 4.2 “Communication in the event of a connection failure”), it must be ensured that first an attempt to read the value of a register or binary is made and only in case of successful communication subsequent communications of read and write requests are made.

On the basis of the above information, the algorithm created in the previous chapter is modified so that for one Slave station there are two **Case** branches solving the first experimental read communication and in case of a successful TCP connection then the second communication of the remaining registers or binaries is performed. If the TCP connection was not OK, the subsequent behaviour will be controlled by whether it is the first or second broken TCP communication. In case of the first failure, the index of this problematic Slave station is saved for subsequent skipping. In the case of the second failure, the entire communication algorithm is temporarily interrupted. The detection of the TCP connection status is performed by the **MdbmCliSt** module.

The algorithm for operating one Slave station can look like this:

```
// one Slave station is defined by two Case states
// first Case state starting communication with Slave station
Case 0
    If TCP_SkipSl != 0
        // communication should be performed
        // starting the first manual reading of communication with the Slave station
        MdbmRead 10, 1, NONE
        Let @SecondComm = false // this is the first communication with the Slave
station
        Let TCP_CliSt = 0 // zeroing the last TCP communication state
        // time to create TCP connection and send frame
        Let TCP_Wait = TCP_Wait_10s
```

```

        Let TCP_Ndx = 1 // transition to the communication evaluation state
    Else
        // first TCP break - this Slave station is skipped
        Let TCP_Ndx = 2
    EndIf
EndCase

// second Case state verifying that a correct TCP connection has been made
Case 1
    // finding the TCP connection status
    MdbmCliSt 10, TCP_CliSt, NONE
    If (TCP_Wait > 0)
        If TCP_CliSt == 2
            // TCP connection was fine
            If @SecondComm
                // and the remaining communications as also in order
                // - switch to the next Slave station
                Let TCP_Ndx = 2
            Else
                // second communication - all remaining read and write registers
                Let @SecondComm = true
                // Remaining reading
                MdbmMark 10, 3, 1, 25, NONE
                // Event writing
                // DO
                If AMiNi_DO[0,0] != AMiNi_DO_pr[0,0]
                    Let AMiNi_DO_pr[0,0] = AMiNi_DO[0,0]
                    MdbmMark 10, 6, 100, 1, NONE
                EndIf
                // AO
                VarWStat AMiNi1_AO, @AO_w, 0
                If @AO_w
                    MdbmMark 10, 6, 101, 4, NONE
                EndIf
                Let TCP_Wait = TCP_Wait_10s // time for communication
                Let TCP_CliSt = 0
            EndIf
        EndIf
    Else
        // no correct TCP connection was established after 10 s
        // - problem with TCP communication
        If TCP_SkipS1 == -1
            // first TCP break - this Slave station will be skipped
            // saving the problematic Slave station (Case value)
            Let TCP_SkipS1 = 0
            Let TCP_Ndx = 2 // transition to the next Slave station
        Else
            // second TCP break - complete silence for 5 minutes
            Let TCP_Ndx = 102 // a value starting with 100 indicates an error
        EndIf
        // 5 minute timeout setting
        Let TCP_TMO = TCP_TMO_5m
    EndIf
EndCase

```

The algorithm is included in the appendix ap0059_ap_en_xxx.zip. This is a sample application called “modbstcp_p4_en_xxx.dso” created in the DetStudio development environment. This application has been created for the control system **AMiNi4DW2**. However, it can be modified to suit any control system with the letter “W” included in the system name using the DetStudio menu “Tools/Change station”.

6 Control system as Slave

The definition of MODBUS communication in the Slave role is done using a double definition:

- ◆ creating a communication definition of the protocol in the Slave role,
- ◆ definition of data points for communication.

6.1 Communication definition

Creating a communication definition of MODBUS protocol in the Slave role represents inserting a definition of the communication item **ModbusSlave** into the application. The insertion is done in DetStudio in the Project window in the “Project/Communication/Modbus” node. When the context menu above this item is opened, the **Add Slave** item is selected.

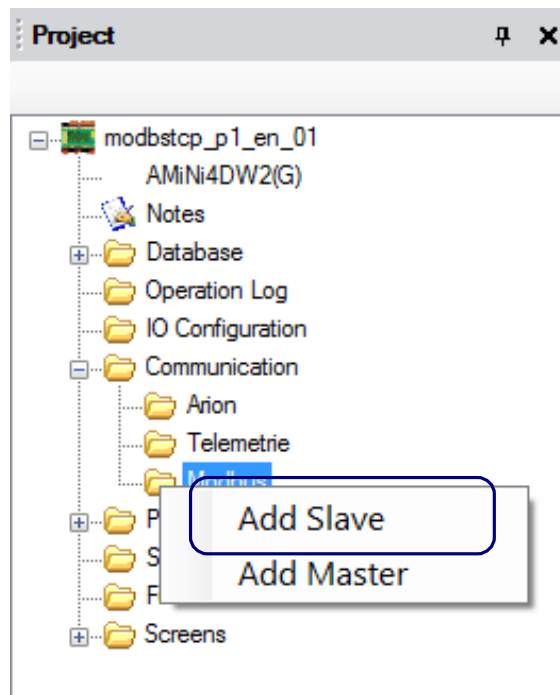


Fig. 11 – Item “Add Slave” in “Modbus” communication definition

After the definition is added, a communication node **ModbusDevice0** is created with default properties values:

- ◆ **Address:** 1
- ◆ **BaudRate:** 19,200
- ◆ **DataBits:** 8
- ◆ **LastError:** NONE
- ◆ **Mode:** SerialLineRTU
- ◆ **Parity:** Even
- ◆ **SerialPort:** 0
- ◆ **StopBit:** One

In order to communicate via MODBUS TCP protocol, it is necessary to set the value of the property **Mode** to “Modbus_TCP”.

More information is available in the Help section of DetStudio called “PseDet – Creating control processes”, in chapter “Contents/Communication/Modbus/Slave – creating and setting general parameters”.

6.2 Definition of data points for communication

After defining the **ModbusX** node, it is possible to double-click on it in the Project window to call up the definition table of communication data points.

In this table, control system variables are defined in individual tabs of data point groups (Holding registers, Input registers, Coils and Discrete inputs); these variables are to be available under selected addresses.

The definition of individual data points can be done e.g. by dragging and dropping from the ToolBox window. More information on definition of data points is available in the Help section of DetStudio called "PseDet – Creating control processes", in chapter "Contents/Communication/Modbus/Slave – table editor".

6.3 Communication statuses

The status of the communication from the Master side is available after assigning the **LastError** property variable in the **ModbusX** communication definition. The expected variable type is I.

It is recommended to use this property especially when debugging the communication, when its value can be used to obtain information about a possible communication error. The assign variable is to take values according to the following table:

Error code	Significance
0	No error.
1	NOS version too low.
2	System timer allocation error.
3	Communication port allocation error.
4	Last frame received had an incorrect check sum.
5	Last frame received had an incorrect length.
6	Last frame received included a request for an unmapped address.
7	Last frame received included a request for an unsupported function.
8	Last frame received required more data than is available for a response frame.
9	Last frame received included incorrect data (function 6 ON, OFF).
10	Too wide space between incoming characters.
11	Error in ASCII reception: – frame too long, – unexpected character (only textual hexa digits must be inside the frame), – no LF followed after CR.
12	The module has not been launched yet (no parameter evaluation and communication port allocation).

6.4 Example of a control system parametrization as Slave

There is a requirement to create an application, after loading which the **AMiNi4DW2** control system will work as a remote input and output module communicating via MODBUS TCP protocol. At the same time, the application is to be universal enough to be able to communicate analogue values in a decimal or integer form.

1. Select the IP address of the control system. E.g. 192.168.1.2.
2. Create variables according to the following table.

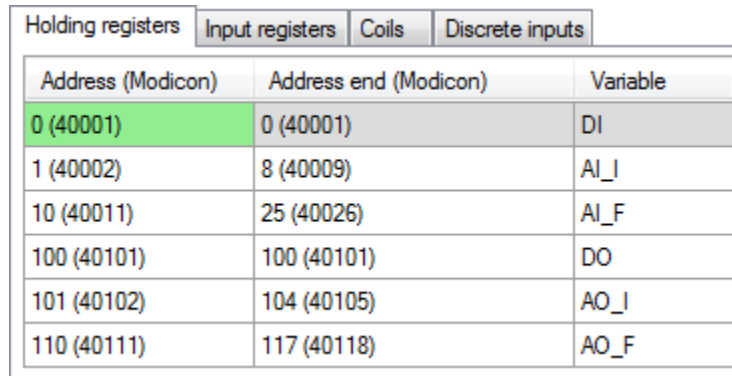
Variable	Type	Comment
DI	I	Digital inputs.
AI_I	MI[1,8]	Integer values of analogue inputs.
AI_F	MF[1,8]	Float values of analogue inputs.
DO	I	Digital outputs.

Variable	Type	Comment
AO_I	MI[1,4]	Integer values of analogue outputs.
AO_F	MF[1,4]	Float values of analogue outputs.
tempF	F	Auxiliary Float variable.
Mdbs_Err	I	Code of the previous communication error.

3. Select the addresses and types of data points to represent the variables according to the following table.

Variable	Address	Data point type
DI	0	Holding register
AI_I	1 to 8	Holding register
AI_F	10 to 25	Holding register
DO	100	Holding register
AO_I	101 to 104	Holding register
AO_F	110 to 117	Holding register

4. Based on the previous table, define the definition rows in the **Modbus0** node.



Address (Modicon)	Address end (Modicon)	Variable
0 (40001)	0 (40001)	DI
1 (40002)	8 (40009)	AI_I
10 (40011)	25 (40026)	AI_F
100 (40101)	100 (40101)	DO
101 (40102)	104 (40105)	AO_I
110 (40111)	117 (40118)	AO_F

Fig. 12 – Definition of MODBUS Slave data points and assigned variables

5. In a periodic process, we create an algorithm that will read inputs and write outputs based on the values of variables. The code may look as follows:

```
// ----- DI -----
DigIn #0, DI, 0x0000

// ----- AI -----

// AI0 and AI1 as Ni1000 / 6,180 ppm
// Temperature 12.45 °C corresponds to value 1245 in Int register
Ni1000 #Ni10001_0, AI_F[0,0], 6180
Let AI_I[0,0] = Int(AI_F[0,0] * 100)

Ni1000 #Ni10001_1, AI_F[0,1], 6180
Let AI_I[0,1] = Int(AI_F[0,1] * 100)

// AI2 and AI3 as Pt1000 / 3,850 ppm
// Temperature 12.45 °C corresponds to value 1245 in Int register
Pt1000 #Ni10001_2, AI_F[0,2], 3850
Let AI_I[0,2] = Int(AI_F[0,2] * 100)

Pt1000 #Ni10001_3, AI_F[0,3], 3850
Let AI_I[0,3] = Int(AI_F[0,3] * 100)

// AI4 and AI5 for measurement of voltage 0 to 10 V
```



```
// Value 3.678 V corresponds to value 3678 in Int register
AnIn #AI00_4, AI_F[0,4], 10.000, 0.000, 10.000, 0.000, 10.000
Let AI_I[0,4] = Int(AI_F[0,4] * 1000)

AnIn #AI00_5, AI_F[0,5], 10.000, 0.000, 10.000, 0.000, 10.000
Let AI_I[0,5] = Int(AI_F[0,5] * 1000)

// AI6 and AI7 for measurement of current 0(4) to 20 mA
// Value 15.345 mA corresponds to value 15345 in Int register
AnIn #AI00_6, AI_F[0,6], 20.000, 0.000, 20.000, 0.000, 20.000
Let AI_I[0,6] = Int(AI_F[0,6] * 1000)

AnIn #AI00_7, AI_F[0,7], 20.000, 0.000, 20.000, 0.000, 20.000
Let AI_I[0,7] = Int(AI_F[0,7] * 1000)

// ----- DO -----
DigOut DO, #0, 0x0000

// ----- AO -----

// AO0 to AO3 with output 0 to 10 V
// Setpoint value 2.456 V must be written in Int register as value 2456
// WEeither use Float registers 110-111 to 116-117 or Int registers 101 to 104
Let tempF = If(AO_F[0,0] == 0, AO_I[0,0] / 1000, AO_F[0,0])
AnOut #AO00_0, tempF, 10.000, 0.000, 10.000, 0.000, 10.000

Let tempF = If(AO_F[0,1] == 0, AO_I[0,1] / 1000, AO_F[0,1])
AnOut #AO00_1, tempF, 10.000, 0.000, 10.000, 0.000, 10.000

Let tempF = If(AO_F[0,2] == 0, AO_I[0,2] / 1000, AO_F[0,2])
AnOut #AO00_2, tempF, 10.000, 0.000, 10.000, 0.000, 10.000

Let tempF = If(AO_F[0,3] == 0, AO_I[0,3] / 1000, AO_F[0,3])
AnOut #AO00_3, tempF, 10.000, 0.000, 10.000, 0.000, 10.000
```

The algorithm is included in the appendix ap0059_ap_en_xxx.zip. This is a sample application called "modbstcp_p4_en_xxx.dso" created in the DetStudio development environment. This application has been created for the control system **AMiNi4DW2**. However, it can be modified to suit any control system with the letter "W" included in the system name using the DetStudio menu "Tools/Change station".

7 Appendix A

7.1 Compatibility with communication initialization via modules

7.1.1 MODBUS Master

It is not possible to define MODBUS TCP Master communication using modules in the initialization or periodic process.

7.1.2 MODBUS Slave

`MODBS_Var` and `MODBS_IS1` modules can also be used to initialize MODBUS TCP Slave communication, which are typically placed in the initialization process.

However, in terms of internal functionality, it is an identical communication to a table definition. For this reason, it **is possible** to combine both communication definitions on the same COM interface.

Find out more in the Help section of DetStudio called “PseDet – Creating control processes”, in chapter “Contents/Communication/Modbus” in the section “Backward compatibility”.

8 Technical support

For all information regarding the communication in a network of the MODBUS TCP in AMiT control systems, please contact AMiT Technical Support. The Technical support is best contacted via e-mail at **support@amit.cz**.

9 Warning

In this document, AMiT, spol. s r. o. provides information as it is, and the company does not provide any warranty concerning the contents of this publication and reserves the right to change the documentation content without any obligation to inform anyone or any authority.

This document can be copied and redistributed under the following conditions:

1. The whole text (all pages) must be copied without making any modifications.
2. All redistributed copies must retain the AMiT, spol. s r. o. copyright notice and any other notices contained in the documentation.
3. This document must not be distributed for profit.

The names of products and companies used herein may be trademarks or registered trademarks of their respective owners.