Library for modbusTCP Master and Slave

Version 2.00 for Automation Studio 4.x

Date: 7.11.2022

Contents

1 Introduction

The library “MbusTCP” provides support for modbusTCP master and slave on all B&R targets that provide an Ethernet interface. Remember that the modbusTCP **master** is also called **client** and the **slave** is called **server**. All sample code in this documentation is written in Structure Text and marked with a gray frame.

The following function codes are supported

Read coils	Write single coil
Read discrete inputs	Write single register
Read holding register	Write multiple coils
Read input registers	Write multiple registers

1.1 System requirements

This document was made for Automation Studio 4.x. It was tested with Automation Studio 4.9 but should also work with lower versions.

This library supports all B&R targets that provide an Ethernet interface. The preferred task class should be the IDLE task (Task class #8 on SG4, Task class #4 on SGC)

2 Adding the library

Copy and paste the folder modbusTCP from the sample project to your project. When you copy the folder in Automation Studio all depending libraries should be added as well.

3 ModbusTCP Master

When you add the master function block you see the following parameters.

```
MBmaster_0(enable := , device := , port := , slave_ip_addr := , p_cfg := , p_log :=);
```

- **enable**
  The function block is only executed when enable is 1
- **device**
  Device string for the Ethernet interface (see 5.5). If "0" is specified, then "listening in" takes place on all interfaces.
- **port**
  Ethernet port. Typical 502 for modbusTCP
- **slave_ip_addr**
  This is the IP address of the modbusTCP slave
- **p_cfg**
  Pointer to configuration structure. This is a variable of type modbus_master_cfg_typ that defines the request actions that are sent to the slave. The master supports up to 20 separate action configurations per slave. Each action has the following parameters. If you use COPY-Mode to transfer your application this pointer has to initialized in the cyclic part of the task.
  - **cyclic/single**
    Defines if the action is called in cyclic or only once
  - **unit**
    The unit is ignored by most devices. Some devices may still use this value for compatibility reasons with modbusRTU.
  - **p_pv**
    Pointer to a variable where the read data is stored or the write data is transferred from
  - **quantity**
    Number of words to read/write. The Modbus specification only allows a payload size of 253 bytes. Therefore the maximum quantity for analog registers is 125 and for coils 2000.
  - **start_addr**
    Start address on the slave side from where to read/write
  - **timer**
    Cycle time for recurring actions
  - **type**
    ModbusTCP function type (see 5.5)
- **p_log**
  The logger provides diagnostic information and action results. It consists of a string array with LOG_LINE_NUM entries that each have a length of LOG_LINE_LEN.
- **last_error**
  Shows the latest error number (see 6.1)
- **my_ip**
  Shows the device IP address
- **status**
  Shows the current function block status

Sample variable declaration

```
VAR
          ModbusTcpMaster : MBmaster;
          MasterConfig : modbus_master_cfg_typ := (0);
END_VAR
(*IO sample arrays*)
VAR
          A_Out : ARRAY[0..20] OF UINT := [21(0)];
          A_In : ARRAY[0..20] OF UINT := [21(0)];
          D_Out : ARRAY[0..49] OF BOOL := [50(FALSE)];
          D_In : ARRAY[0..49] OF BOOL := [50(FALSE)];
END_VAR
(*MISC variables*)
VAR
          ip : STRING[20];
          logger : ARRAY[0..19] OF STRING[50];
          StartStop : BOOL := FALSE;
END_VAR
```

Sample init program with 4 action items.

```
PROGRAM _INIT
          strcpy(ADR(ip), ADR('192.168.0.30'));                    (* Slave device IP address *)

          (* ---------------------------------------------------------------------
            Sample for reading/writing data from/to a modbusTCP slave
            --------------------------------------------------------------------- *)
          MasterConfig.action_enable[0].cyclic   := 1;                        (* Read data cyclic *)
          MasterConfig.action_param[0].p_pv      := ADR(D_In);       (* Store data in this variable *)
          MasterConfig.action_param[0].quantity  := 20;                       (* Number of items to read *)
          MasterConfig.action_param[0].unit      := 1;                        (* Unit address - ignore if not specified *)
          MasterConfig.action_param[0].start_addr := 16#0;           (* Read from this slave address *)
          MasterConfig.action_param[0].timer     := 500;                      (* Refresh timer for this item *)
          MasterConfig.action_param[0].type      := 2;                        (* Function code for this item (2 := read
discrete inputs *)

          MasterConfig.action_enable[1].cyclic   := 1;                        (* Write data cyclic *)
          MasterConfig.action_param[1].p_pv      := ADR(D_Out);      (* Take data from this address *)
          MasterConfig.action_param[1].quantity  := 20;                       (* Number of items to write *)
          MasterConfig.action_param[1].unit      := 1;                        (* Unit address - ignore if not specified *)
          MasterConfig.action_param[1].start_addr := 16#0;           (* Write to this slave address *)
          MasterConfig.action_param[1].timer     := 500;             (* Refresh timer for this item *)
          MasterConfig.action_param[1].type      := 15;                       (* Function code for this item (16 := write
multiple register) *)

          MasterConfig.action_enable[2].cyclic   := 1;                        (* Read data cyclic *)
          MasterConfig.action_param[2].p_pv      := ADR(A_In);       (* Store data in this variable *)
          MasterConfig.action_param[2].quantity  := 20;                       (* Number of items to read *)
          MasterConfig.action_param[2].unit      := 1;                        (* Unit address - ignore if not specified *)
          MasterConfig.action_param[2].start_addr := 16#0;           (* Read from this slave address *)
          MasterConfig.action_param[2].timer     := 500;             (* Refresh timer for this item *)
          MasterConfig.action_param[2].type      := 4;                        (* Function code for this item (2 := read
discrete inputs) *)

          MasterConfig.action_enable[3].cyclic   := 1;                        (* Write data cyclic *)
          MasterConfig.action_param[3].p_pv      := ADR(A_Out);      (* Take data from this address *)
          MasterConfig.action_param[3].quantity  := 20;                       (* Number of items to write *)
          MasterConfig.action_param[3].unit      := 1;                        (* Unit address - ignore if not specified *)
          MasterConfig.action_param[3].start_addr := 16#0;           (* Write to this slave address *)
          MasterConfig.action_param[3].timer     := 500;             (* Refresh timer for this item *)
          MasterConfig.action_param[3].type      := 16;                       (* Function code for this item (16 := write
multiple register) *)
          (* ---------------------------------------------------------------------
            Sample for reading data from a modbusTCP slave
            --------------------------------------------------------------------- *)
          ModbusTcpMaster.enable                 := 1;               (* Start master *)
          ModbusTcpMaster.device                 := ADR('IF5');      (* Ethernet device string *)
          ModbusTcpMaster.slave_ip_addr          := ip;              (* Transfer slave device IP address *)
          ModbusTcpMaster.p_cfg                  := ADR(MasterConfig);        (* Pointer to configuration *)
          ModbusTcpMaster.p_log                  := ADR(logger);              (* Pointer to logger string array *)

          StartStop := 1;
END_PROGRAM
```

The cyclic part of the task only contains the function call

```
PROGRAM _CYCLIC
          (* ---------------------------------------------------------------------
            Cyclic function call
            --------------------------------------------------------------------- *)
          IF(StartStop) THEN
                    ModbusTcpMaster();
          END_IF
END_PROGRAM
```

4 ModbusTCP Slave

The slave supports up to 3 simultaneous connections a time. By default, the slave has 256 addresses for digital and analog input and output data. The following parameters are part of the function call.

```
MBslave_0(enable := , device := , port := , p_cfg := , p_log := , master_timeout := );
```

- **enable**
  The function block is only executed when enable is 1
- **device**
  Device string for the Ethernet interface (see 5.4)
- **port**
  Ethernet port. Typical 502 for modbusTCP
- **p_cfg**
  Pointer to configuration structure. This is a variable of type modbus_slave_cfg_typ that defines memory areas for analog and digital registers. The slave supports 100 addresses for each register by default.
  - **p_coils[0..255]**
    Defines the addresses for the digital outputs
  - **p_discrete_inputs[0..255]**
    Defines the addresses for the digital inputs
  - **p_holding_registers[0..255]**
    Defines the addresses for the analog outputs
  - **p_input_registers[0..255]**
    Defines the addresses for the analog inputs
- **p_log**
  The logger provides diagnostic information and action results. It consists of a string array with LOG_LINE_NUM entries that each have a length of LOG_LINE_LEN.
- **master_timeout**
  The slave expects a request from the master every X ms to verify that the master is still up and running. When the master times out the slave will close the connection. A value of 0 will disable this function and the slave will always keep the connection open until the master disconnects. It is highly recommend to use this feature because the slave could run out of slots when the connection between master and slave is interrupted without a proper disconnect.
- **last_error**
  Shows the latest error number (see 6.1)
- **my_ip**
  Shows the device IP address
- **status**
  Shows the current function block status

Sample variable declaration

```
VAR
        ModbusTcpSlave : MBslave;
        SlaveConfig : modbus_slave_cfg_typ := (0);
END_VAR
(*IO sample arrays*)
VAR
        DO_Bool : ARRAY[0..255] OF BOOL := [256(FALSE)];
        DI_Bool : ARRAY[0..255] OF BOOL := [256(FALSE)];
        AO_Word : ARRAY[0..255] OF UINT := [256(0)];
        AI_Word : ARRAY[0..255] OF UINT := [256(0)];END_VAR
(*MISC variables*)
VAR
        interface : STRING[20];
        idx : UINT := 0;
        logger : ARRAY[0..19] OF STRING[50];
        StartStop : BOOL := FALSE;
END_VAR
```

Sample init program with 4 action items.

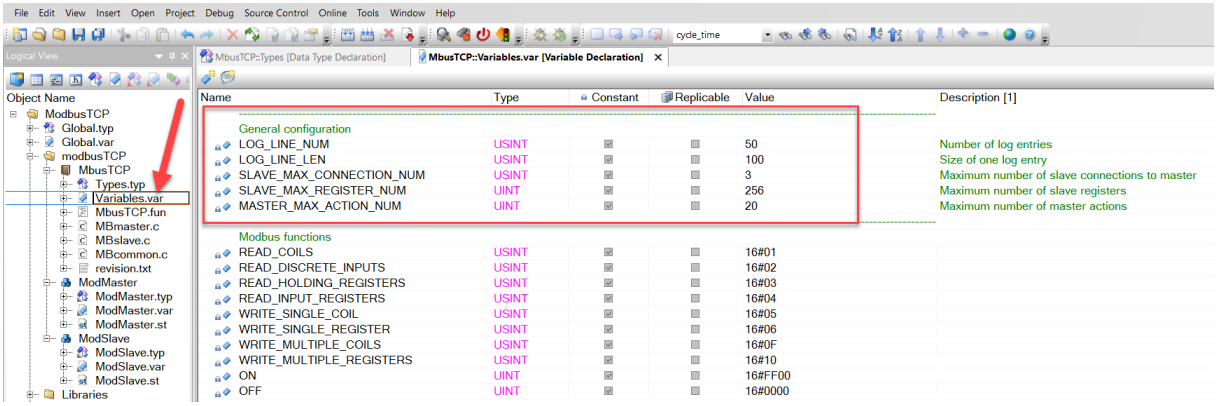
```
PROGRAM _INIT
        brsstrcpy(ADR(interface), ADR('IF5'));                                  (* Ethernet interface used on
this PLC *)

        (* ---------------------------------------------------------------------
          Reset function blocks and data structure on startup (SGC)
          --------------------------------------------------------------------- *)
    brsmemset(ADR(ModbusTcpSlave), 0, SIZEOF(ModbusTcpSlave));
    brsmemset(ADR(SlaveConfig), 0, SIZEOF(SlaveConfig));
    brsmemset(ADR(logger), 0, SIZEOF(logger));

        (* ---------------------------------------------------------------------
          Create default IO mapping FOR digital AND analog area
          --------------------------------------------------------------------- *)
        FOR idx:=0 TO SIZEOF(DO_Bool)-1 DO
                SlaveConfig.p_coils[idx] := ADR(DO_Bool[idx]);
        END_FOR;
        FOR idx:=0 TO SIZEOF(DI_Bool)-1 DO
                SlaveConfig.p_discrete_inputs[idx]     := ADR(DI_Bool[idx]);
        END_FOR;
        FOR idx:=0 TO SIZEOF(AO_Word)/SIZEOF(AO_Word[0])-1 DO
                SlaveConfig.p_holding_registers[idx] := ADR(AO_Word[idx]);
        END_FOR;
        FOR idx:=0 TO SIZEOF(AI_Word)/SIZEOF(AI_Word[0])-1 DO
                SlaveConfig.p_input_registers[idx]     := ADR(AI_Word[idx]);
        END_FOR;

        (* ---------------------------------------------------------------------
          Configure FUNCTION block instance
          --------------------------------------------------------------------- *)
        ModbusTcpSlave.enable        := 1;                      (* Enable function block *)
        ModbusTcpSlave.device        := ADR(interface);         (* Ethernet device string *)
        ModbusTcpSlave.p_cfg         := ADR(SlaveConfig);       (* IO configuration *)
        ModbusTcpSlave.p_log         := ADR(logger);            (* Pointer to logger string array
        ModbusTcpSlave.master_timeout := 60000;                 (* Disconnect master after x milliseconds *)

        StartStop := 1;
END_PROGRAM
```

The cyclic part of the task only contains the function call

```
PROGRAM _CYCLIC
        (* ---------------------------------------------------------------------
          Cyclic function call
          --------------------------------------------------------------------- *)
        IF(StartStop) THEN
                ModbusTcpSlave();
        END_IF
END_PROGRAM
```

5 Tipps and Hints

The library has a few parameters that can be adjusted by the user. Remember that this value affects the response time for request. Make a complete rebuilt before downloading the project. Make a complete rebuilt before downloading the project.



5.1 Slave: How to increase the number of simultaneous connections for the slave?

The slave supports 3 simultaneous connections meaning that 3 masters can connect to the slave at the same time. To increase this value change the constant SLAVE_MAX_CONNECTION_NUM.

5.2 Slave: How to increase the number of registers?

The slave supports 256 entries for analog/digital input and output data by default. To increase this value change the constant SLAVE_MAX_REGISTER_NUM.

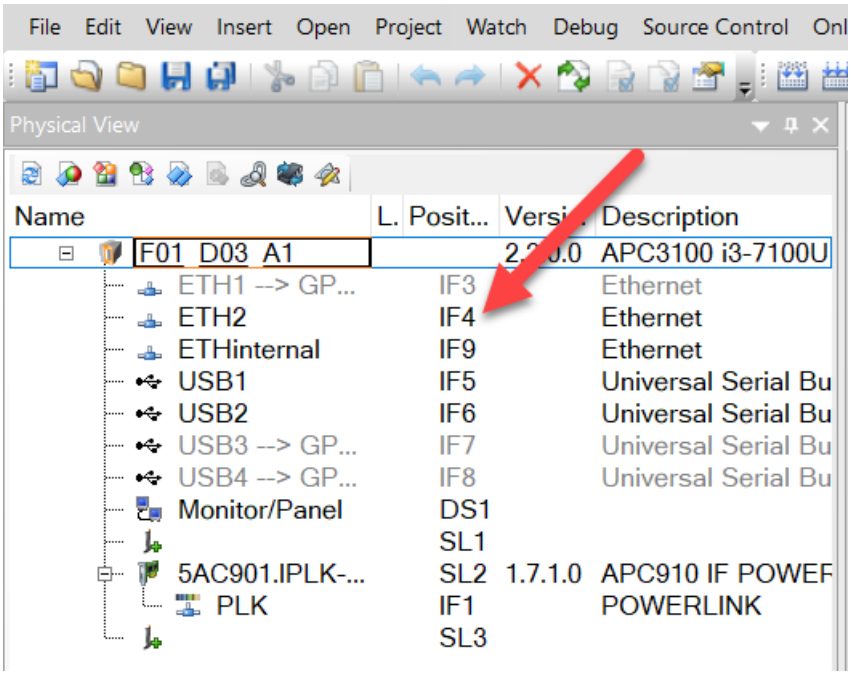
5.3 How to increase the number of actions?

The master supports 20 read or write actions per slave connection. To increase this value change the constant MASTER_MAX_ACTION_NUM.

5.4 How to find the Ethernet device string

The following procedure describes how to find the device string that is needed for the master and slave configuration.

1. Click on the “Physical View” at the bottom
2. The column position contains the device name



5.5 ModbusTCP function codes

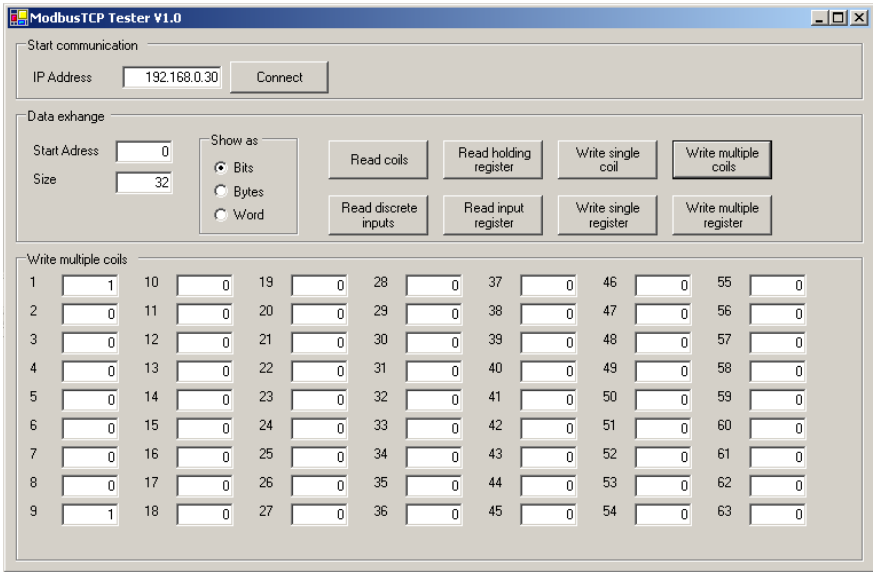
- Different basic types for data model
  - Coils (DO)
  - Discrete Inputs (DI)
  - Holding Register (AO)
  - Input Register (AI)

Function code	Protocol-specific ID	Description	Method
1	Read coils	Read multiple digital outputs	As bit
2	Read discrete inputs	Read multiple digital inputs	As bit
3	Read holding registers	Read multiple analog outputs	As word
4	Read input register	Read multiple analog inputs	As word
5	Write single coil	Write to a digital output	As bit
6	Write single register	Write to an analog output	As word
15	Write multiple coils	Write multiple digital outputs	As bit
16	Write multiple registers	Write multiple analog outputs	As word
23	Read/write multiple registers	Read and write several analog outputs in one command	As word

5.6 Windows Test Master

This package comes with a Windows test master application. This is a basic version of a modbusTCP master that can be used with the library slave application. It requires Microsoft .Net Framework 2.0. The latest version can also be downloaded on GitHub

stephan1827/modbusTCP-DotNET: Visual Studio modbusTCP class (github.com)



6 6 Appendix

6.1 6.1 Error numbers and text

COM → Common error

Slave → modbusTCP slave error

Master → modbusTCP master error

50000 COM: Critical error, disable and re-enable FUB

50001 COM: Frame message is to short
Solution: If this is a onetime event than it is just a broken Ethernet frame. If it happens permanently the other device sends wrong information.

50002 COM: Message format is incorrect
Solution: If this is a onetime event than it is just a broken Ethernet frame. If it happens permanently the other device sends wrong information.

50003 COM: Receive buffer exceeded maximum configured size
Solution: The message is to long. Increase the receive buffer (see 5.2 and 5.3).

50004 COM: Variable address is empty
Solution: One or multiple addresses have no variable assigned. Make sure that every address that is configured has a valid pointer.
If running master check MasterConfig.action_param[].p_pv
If running slave check SlaveConfig.p_coils[], SlaveConfig.p_discrete_inputs[], SlaveConfig.p_holding_registers[], SlaveConfig.p_input_registers[]

50006 COM: Quantity is higher than the Modbus limit
Solution: MasterConfig.action_param[].quantity must not be higher than 125 for analog values and 2000 for digital.

50100 Slave: More than MAX_MASTER are trying to connect
Solution: More masters than configured try to connect (see 5.1).

50101 Slave: Address exceeded maximum configured size
Solution: The master is trying to access a registers address that is higher than the maximum. See 5.2 on how to increase the number of registers.

50102 Slave: Size exceeded maximum configured size
Solution: The master is trying to access a registers quantity that is higher than the maximum. See 5.2 on how to increase the number of registers. Another reason is that the master is trying to access more data than the Modbus limit in one request (125 for analog/2000 for digital).

50103 Slave: Function code is not supported
Solution: The master calls a function code that is not supported. See 1.1 for the supported function codes.

50104 Slave: Master timed out
Solution: The master did not send at least request within the maximum timeout (master_timeout). Change timeout to 0 or call cyclic request from master.

50110 Slave: Configuration error in num masters
Solution: The arrays under 5.1 have different sizes. Makes sure all arrays have the same number of elements.

50200 Master: Bad data received from slave
Solution: If this is a one-time event than it is just a broken Ethernet frame. If it happens permanently the other device sends wrong information.

50201 Master: Data received from slave too short
Solution: If this is a one-time event than it is just a broken Ethernet frame. If it happens permanently the other device sends wrong information.

50202 Master: Data received from slave too long
Solution: The maximum size for a frame is 500 bytes which limits the amount of analog registers to 240 for one request. To solve this issue break up the request into multiple commands with 240 registers each. As an alternative the send and receive buffer of the library can be increased in the structure modbus_master_internal_typ.

50210 Master: A request frame to the slave timed out
Solution: The slave did not respond on a request within the 5 seconds. This can be connection problem.

50211 Master: Connection to the slave timed out
Solution: Check the slave IP address.

50300 Master: Invalid Ethernet device
Solution: Check parameter ModbusTcpMaster.device and make sure it matches the Ethernet interface.

6.2 Revision History

Version 2.0
CHANGE: Ported library to AS4
CHANGE: Added device configuration error to slave
CHANGE: Changed logger structure to pointer with fixed size
CHANGE: Added error message to port open steps
CHANGE: Auto detect cycle time in master
CHANGE: Added configuration constants
CHANGE: Updated configuration