RUNMODE xCP MODBUS

SOFTWARE COMMUNICATION DRIVERS FOR SIEMENS S7-300/-400 PLC

MODBUS RTU SLAVE V1.1

CP-independent software driver suitable for

- any S7-compatible serial port communication processor (CP), either rack mounted or Profibus networked
- any S7-compatible PtP CPU
- TCP tunneling using Ethernet CPs

Documentation last update: June 21, 2012

RUNMODE
Industrial Automation Software
Via C. B. Cavour, 7
31040 Volpago del Montello (TV)
ITALY
www.runmode.com

Index

RUNMODE S7 xCP MODBUS RTU SLAVE features

The RUNMODE S7 xCP MODBUS RTU driver can be used in conjunction with any CP that provides a plain ASCII communication link, such are CP340/440, CP341/441, ET200S series serial modules, third-party modules modules (e.g. Helmholz SAS340, Vipa CPs, Wago 750 series, and others).

The driver supports the RTU version of Modbus. It manages the protocol telegrams but does not actually send nor receive serial data. Proper interface flags and data areas are provided thus allowing the PLC programmer to implement the necessary communication block calls according to the specific CP model and manufacturer brand.

Memory areas in the PLC

Due to the different memory model of Modicon and Simatic systems, MODBUS registers will be actually mapped in the Simatic PLC as a data block area. The data block solution as intermediate data storage allows also the adaptation of incoming and outgoing data (e.g. solve endian issues of word-sized analog inputs values or double-word float variables).

The driver allows the indication of different datablocks for registers 3xxxx and for registers 4xxxx. While the original MODBUS standard states a range of 1 to 9999 registers per area, the Runmode xCP driver does not check for that limit, thus allowing an extended range up to 32767 16-bit registers (64KB) or, in any case, up to the current length of the data blocks.

Discrete coils addressing refers nominally to memory flags (M) area, but an option allows either to redirect the coils to a “coils data block” or to the holding registers data block. In the latter case, individual bit access is then possible even within the holding registers, thus offering a better interface with HMI/SCADA systems.

In case the true memory flags area is selected, during the initialization procedure the driver detects automatically the extent of the flags area by reading the CPU system data.

Coils and registers data blocks length is also detected in order to prevent addressing of variables beyond the data blocks extent.

Implemented functions

The Runmode xCP SLAVE driver provides the following set of MODBUS functions:

- FC01 read discrete coils
- FC03 read holding registers
- FC04 read input registers
- FC05 write single coil
- FC06 write single holding register
- FC16 write multiple holding registers
- FC08 diagnostics, limited to the loopback "Return Query Data" function only

Implemented exceptions messages

Some exception response messages are also implemented in the S7 block, so that the driver, according to the MODBUS standard, will react with proper response messages to most common exceptions.
The messages are limited to the following exceptions:

Exception code	Name	description
01	Illegal function	The function code received in the query is not an allowable action for the slave. This may be because the function code is not implemented in the unit selected.
02	Illegal data address	The data address received in the query is not an allowable address for the server (or slave). More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed, while a request with offset 96 and length 5 will generate exception 02.

PLC memory footprint

The driver is made of just one code block, namely FB100, which can be renamed.

The driver need an instance DB for FB100 and it shares the CP send/receive mailbox data blocks.

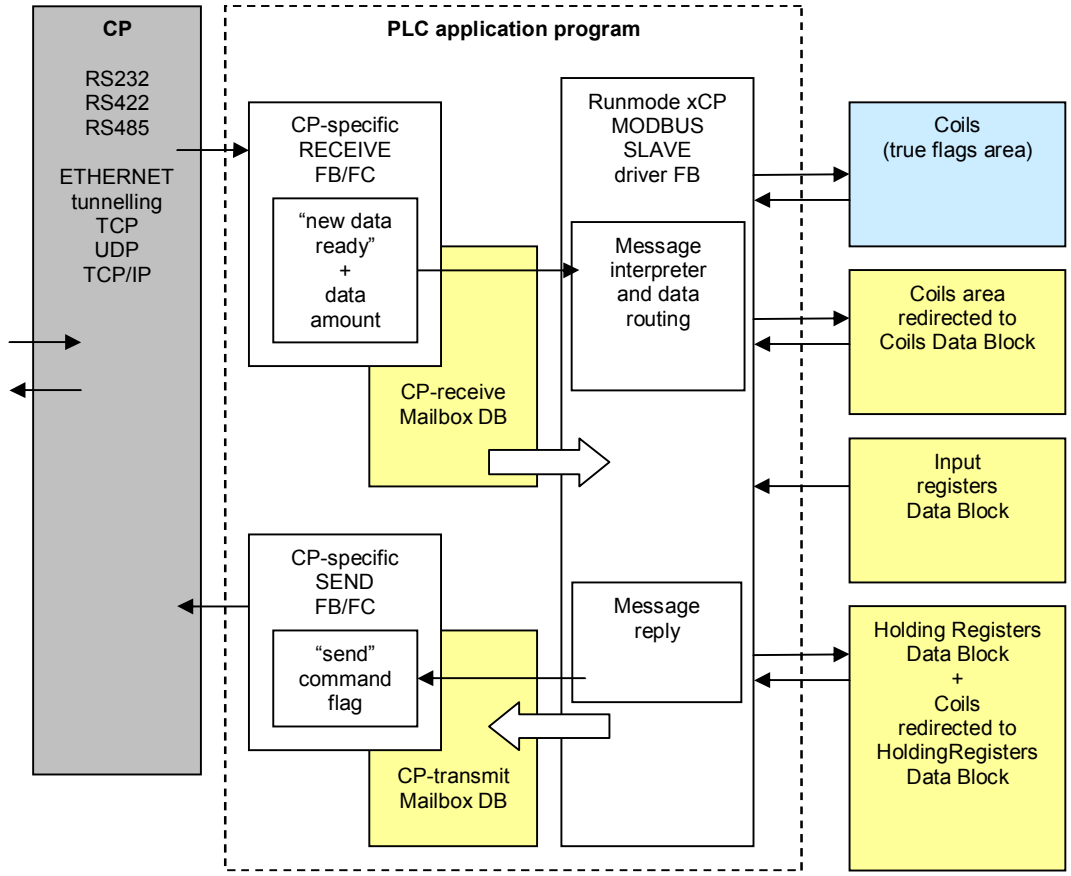
The size of the mailboxes should be at least 260 bytes each.
In case the exchanged messages are proven to be shorter and there is need to save as much memory as possible, the mailbox DBs size might be reduced accordingly.

Block	description	Local data (bytes)	MC7 - machine code S7 (bytes)	Load memory (bytes)	Work memory (bytes)
FB100	driver	130	3510	3936	3546
DB	FB100 instance DB	- - -	170	402	206
DB	send mailbox	- - -	depends on max amount of data to be exchanged	depends on max amount of data to be exchanged	depends on max amount of data to be exchanged
DB	receive mailbox	- - -	depends on max amount of data to be exchanged	depends on max amount of data to be exchanged	depends on max amount of data to be exchanged

How the driver works

In detail:

1. Once a message has been received from the CP, the application program must forward the amount of received data and reception flag to the MODBUS driver.
2. The MODBUS driver checks continuously the “start” flag. When “true”, the flag is immediately reset internally and the driver begins the telegram processing: it checks for the valid contents of the received data and takes related actions according to the read or write request. Data is then directly read from or written to the related memory areas (e.g. holding registers DB).
3. The MODBUS driver prepares the acknowledge telegram to be sent back to the master. The driver writes data directly into the “transmit mailbox” DB.
4. The MODBUS driver provides a “telegram ready” flag along with the indication of how many bytes must be sent (“data amount”).
5. The application program must check the state of the “send request” flag and manage the related CP send request. The “send request” flag must be reset by the user program.



Note: The data is read, checked, transferred and response prepared within the same PLC scan.

Functions details

Coils

- FC01 - read discrete coils: allows the master to read a number of coils in the slave memory.
- FC05 - write single coil: allows the master to write an individual coil in the slave memory.

The coils function will operate on the memory area defined in the setup section:

- True memory flags (Marker) area
- A dedicated "coils data block"
- Share the holding registers data block

NOTE: Due to specific PLC instructions used, function FC01 implementation in the current driver is CPU time consuming. Therefore the maximum amount of coils that can be accessed for each request **is limited to 16 elements**.

Input registers

- FC04 – read input registers: reads read-only registers in the PLC memory, commonly defined as 3xxxxxx registers at MODBUS master side.

Holding registers

- FC03 - read holding registers: reads read/write registers in the PLC memory, commonly defined as 4xxxxxx registers area at MODBUS master side.
- FC06 - write single holding register: writes an individual register in the PLC memory, commonly defined as 4xxxxxx registers area at MODBUS master side.
- FC16 - write multiple holding registers: writes a contiguous series of registers in the PLC memory, commonly defined as 4xxxxxx registers area at MODBUS master side.

Diagnostics

- FC08 - diagnostics: the diagnostics is limited to the loopback "Return Query Data" sub-function only. The "Return Query Data" sends back the message just received from the master and it may be used to safely test the communication without accessing to any coils or register areas.

Integrate the xCP driver in your S7 program

Check for correct CP-CPU communication blocks

IMPORTANT: different CPs and CPUs may use own S7 communication blocks for handling the data traffic between CP and CPU (e.g. "P_RCV_RK", "P_SND_RK"), so before integrating the xCP driver make sure you loaded the correct blocks into your S7 program. Check your CPU and CP card documentation.

Resources

The Runmode xCP driver needs the following PLC resources:

- 1 FB block
- 1 instance data block

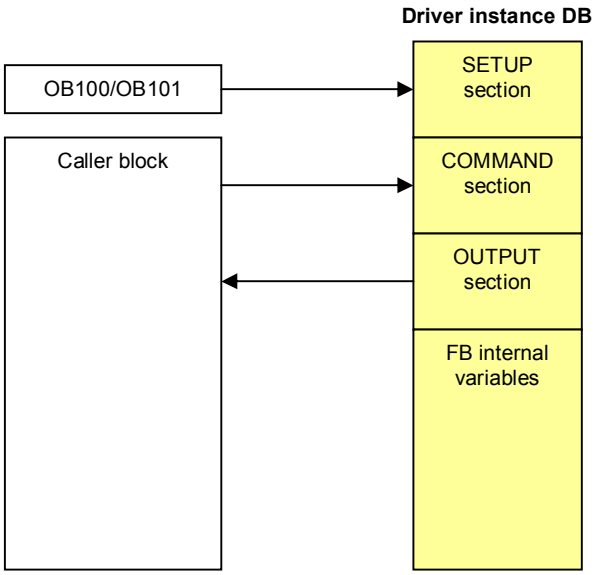
Your own S7 project must also provide some shared resources:

- 1 RX mailbox DB (your CP receive area).
- 1 TX mailbox DB (your CP transmit area).
- 1 DB for allocation of coils (actual DB size may vary according to your application and PLC technical data). The data block is necessary only if coils are enabled and redirected to DB area in the driver setup.
- 1 DB for allocation of input registers (actual DB size may vary according to your application and PLC technical data). The data block is necessary only if input registers are enabled in the driver setup.
- 1 DB for allocation of holding registers (actual DB size may vary according to your application and PLC technical data). The data block is necessary only if holding registers are enabled in the driver setup.

Interface design

In order to minimize its memory footprint, the driver FB has no external interface parameters: all the data must be written directly to the instance DB.
To ease the assignments, the instance DB interface is divided into sections:

- Setup section: contains parameters to be set just once, preferably at PLC startup.
- Command section: input commands to the driver are here located.
- Output section: output commands and data from the driver are here located.



Setup and parameterization

Assuming that DBxx is the driver instance DB, the following is a list of commands and parameters needed to set up the xCP driver.

Setup parameters interface

Assign the necessary setup data in OB100/OB101 as follows:

Setup parameter	Range	Description
DBxx.setup.SlaveNr	1 to 255	Node number of your MODBUS slave
DBxx.setup.RXmailboxDBnr	1 to max allowable by PLC	Number of the data block where incoming data from your CP is stored.
DBxx.setup.TXmailboxDBnr	1 to max allowable by PLC	Number of the data block where outgoing data to your CP is stored.
DBxx.setup.EnableCoils	False / True	Enable coils memory area and related functions. If the area is not enabled all related functions, if requested by the Modbus master, will return exception 01 “Illegal function”.
DBxx.setup.EnableInputRegisters	False / True	Enable input registers memory area and related functions. If the area is not enabled all related functions, if requested by the Modbus master, will return exception 01 “Illegal function”.
DBxx.setup.EnableHoldRegisters	False / True	Enable holding registers memory area and related functions. If the area is not enabled all related functions, if requested by the Modbus master, will return exception 01 “Illegal function”.
DBxx.setup.CoilsRedirection	Char M, D, H	Area where coils are redirected to: ‘M’ = memory flags ‘D’ = coils data block ‘H’ = holding registers DB
DBxx.setup.Coils_DBnr	1 to max allowable by PLC	Number of the DB containing coils. Must be assigned only if coils are enabled by Bxx.setup.EnableCoils and coils are redirected to coils DB (DBxx.setup.CoilsRedirection = ‘D’).
DBxx.setup.InputRegisters_DBnr	1 to max allowable by PLC	Number of the DB containing input registers (4xxxx) area. Must be assigned if input registers are enabled by DBxx.setup.EnableInputRegisters.
DBxx.setup.HoldingRegisters_DBnr	1 to max allowable by PLC	Number of the DB containing holding registers (3xxxx) area. Must be assigned if holding registers are enabled by DBxx.setup.EnableHoldRegisters or coils are redirected to holding registers (DBxx.setup.CoilsRedirection = ‘H’).

Input commands interface

Commands are set according to the following table:

Command interface	Range	Description
DBxx.cmd.init	False / True	Reset command. All setup data is recalculated and communication is reset. The flag must be set by the application program at PLC restart and it is reset automatically by the driver.
DBxx.cmd.RxAmount	1 to max extent of RxMailbox DB	Amount of data received by the Communication Processor (e.g. CP340) and stored in the RxMailbox data block.
DBxx.cmd.start	False / True	Start of Modbus telegram interpretation. The flag must be set by the application program as soon as a new data is received by the Communication processor (e.g. CP340) and it is reset automatically by the driver

Output commands interface

Commands are set according to the following table:

Output interface	Range	Description
DBxx.output.DataAmount	1 to max extent of TxMailbox DB	Amount of data, stored in the TxMailbox data block, to be sent by the Communication Processor (e.g. CP340).
DBxx.output.SendRequest	False / True	Data in the Txmailbox DB is ready to be sent. The application program must check this flag to trigger the transmit function of the Communication processor (e.g. CP340). The flag must be reset by the application program.
DBxx.output.ErrorCode	0..65535	It contains the code of detailed protocol errors trapped by the driver. The information is available until a new “start” request is received by the driver. See the errors table in the “troubleshooting” section.

Driver initialization (reset)

By setting the "init" flag, all the internal variables depending on setup data are recalculated and communication is reset. The "init" flag is reset internally by the driver.

Command	Range	Description
DBxx.cmd.init	False / True	request to initialize the software driver

Example of OB100 programming

Assign the slave number

```
L     17
T     "drvSlaveDB".setup.SlaveNr  //Modbus slave ID number
```

Assign the mailbox DBs

```
L     101
T     "drvSlaveDB".setup.RXmailboxDBnr    //number of RX mailbox datablock
                                            (incoming data from CP)
L     102
T     "drvSlaveDB".setup.TXmailboxDBnr    //number of TX mailbox datablock
                                           (outgoing data to CP)
```

Assign the coils and registers DBs

```
L     110
T     "drvSlaveDB".setup.Coils_DBnr    //number of the DB containing coils

L     111
T     "drvSlaveDB".setup.InputRegisters_DBnr    //number of the DB containing
                                                   input registers 300001..39999

L     112
T     "drvSlaveDB".setup.HoldingRegisters_DBnr    //number of the DB containing
                                                  holding registers 400001..49999
```

Select the coils area

```
L     'D'
T     "drvSlaveDB".setup.CoilsRedirection    //area where coils are redirected
                                               to (M=Memory flags, D=coils DB,
                                               H=holding reg DB)
```

Enable the coils and registers areas and related functions

```
SET
S     "drvSlaveDB".setup.EnableCoils    //enable memory areas
S     "drvSlaveDB".setup.EnableInputRegisters
S     "drvSlaveDB".setup.EnableHoldRegisters

S     "drvSlaveDB".cmd.init       //request to initialize the software driver
```

Sample program

Setting up the communication is fairly simple; just implement the following code sections:

1. call your CP receive block.
2. wait for NDR (New data ready) from the CP and forward the received data summary to the xCP MODBUS driver.
3. call xCP MODBUS driver.
4. wait for output data from the driver and forward outgoing data summary to your CP send block.
5. call your CP send block.

Example using a generic serial port CP

1. call your CP receive block

```
      CALL  "P_RCV_RK_OLD" , "DI_P_RCV"
      EN_R    :=TRUE
      R       :="PLCrestart"
      LADDR   :=256                   //CP I/O address
      DB_NO   :=101                   //CP incoming data DB
      DBB_NO  :=0                     //incoming data byte offset
      L_TYP   :=
      L_NO    :=
      L_OFFSET:=
      L_CF_BYT:=
      L_CF_BIT:=
      NDR     :=
      ERROR   :=
      LEN     :=
      STATUS  :=
```

2. wait for NDR (New Data Ready) from CP and forward the received data summary to the xCP MODBUS driver

```
      U     "DI_P_RCV".NDR              //"new data ready" from CP
      R     "DI_P_RCV".NDR
      SPBN  nRx
      L     "DI_P_RCV".LEN              //"amount of bytes received" from CP
      T     "drvSlaveDB".cmd.RxAmount
      SET
      S     "drvSlaveDB".cmd.start      //command for starting MODBUS data processing
nRx:  NOP   0
```

3. call the xCP MODBUS driver

```
      CALL  "drv_slave" , "drvSlaveDB"  //call MODBUS SLAVE driver
```

4. wait for output data from the xCP MODBUS driver and forward outgoing data summary to the CP

```
      U     "drvSlaveDB".output.SendRequest    //output flag from MODBUS driver
      R     "drvSlaveDB".output.SendRequest
      SPBN  nTx
      L     "drvSlaveDB".output.DataA mount
      T     "DI_P_SND".LEN              //"amount of bytes to send" to the CP
      SET
      S     "DI_P_SND".REQ              //set "transmit request" to the CP
nTx:  NOP   0
```

5. call your CP send block

```
      CALL  "P_SND_RK_OLD" , "DI_P_SND"
       SF      :=
       REQ     :=
       R       :="PLCrestart"
       LADDR   :=256                  // CP I/O address
       DB_NO   :=102                  //CP outgoing data DB
       DBB_NO  :=0                    //outgoing data offset
       LEN     :=
       R_CPU_NO:=
       R_TYP   :=
       R_NO    :=
       R_OFFSET:=
       R_CF_BYT:=
       R_CF_BIT:=
       DONE    :=
       ERROR   :=
       STATUS  :=

      U     "DI_P_SND".DONE
      O     "DI_P_SND".ERROR
      R     "DI_P_SND".REQ              //end of send request
```

S7 code example, German instruction set

OB100

```
Network 1: PLC restart flag

      SET
      S     "PLCrestart"                  //PLC restart flag


Network 2: setup MODBUS SLAVE

      L     17
      T     "drvSlaveDB".setup.SlaveNr  //Modbus slave ID number

      L     101
      T     "drvSlaveDB".setup.RXmailboxDBnr    //number of RX mailbox datablock
      L     102
      T     "drvSlaveDB".setup.TXmailboxDBnr    //number of TX mailbox datablock
      L     110
      T     "drvSlaveDB".setup.Coils_DBnr    //number of the DB containing coils
      L     111
      T     "drvSlaveDB".setup.InputRegisters_DBnr    //number of the input regs DB
      L     112
      T     "drvSlaveDB".setup.HoldingRegisters_DBnr  //number of the holding regs DB
      L     'D'
      T     "drvSlaveDB".setup.CoilsRedirection   //area where coils are redirected to
      SET
      S     "drvSlaveDB".setup.EnableCoils        //enable memory areas
      S     "drvSlaveDB".setup.EnableInputRegisters
      S     "drvSlaveDB".setup.EnableHoldRegisters
      S     "drvSlaveDB".cmd.init          //request to initialize the software driver
```

OB1

```
Network 1: call CP and MODBUS blocks
```

```
  CALL  "P_RCV_RK_OLD" , "DI_P_RCV"
        EN_R    :=TRUE
        R       :="PLCrestart"
        LADDR   :=256                   //CP I/O address
        DB_NO   :=101                   //CP incoming data DB
        DBB_NO  :=0                     //incoming data byte offset
        L_TYP   :=
        L_NO    :=
        L_OFFSET:=
        L_CF_BYT:=
        L_CF_BIT:=
        NDR     :=
        ERROR   :=
        LEN     :=
        STATUS  :=

      U     "DI_P_RCV".NDR              //<-- "new data ready" from CP
      R     "DI_P_RCV".NDR
      SPBN  nRx
      L     "DI_P_RCV".LEN              //<-- "amount of bytes received" from CP
      T     "drvSlaveDB".cmd.RxAmount
      SET
      S     "drvSlaveDB".cmd.start      //<-- command to start MODBUS data process
nRx:  NOP   0

      CALL  "drv_slave" , "drvSlaveDB"  //call MODBUS SLAVE driver

      U     "drvSlaveDB".output.SendRequest    //<-- output flag from MODBUS driver
      R     "drvSlaveDB".output.SendRequest
      SPBN  nTx
      L     "drvSlaveDB".output.DataAmount
      T     "DI_P_SND".LEN              //<-- "amount of bytes to send" to the CP
      SET
      S     "DI_P_SND".REQ              //<-- set "transmit request" to the CP
nTx:  NOP   0


      CALL  "P_SND_RK_OLD" , "DI_P_SND"
       SF       :=
       REQ      :=
       R        :="PLCrestart"
       LADDR    :=256                  //CP I/O address
       DB_NO    :=102                  //CP outgoing data DB
       DBB_NO   :=0                    //outgoing data offset
       LEN      :=
       R_CPU_NO:=
       R_TYP    :=
       R_NO     :=
       R_OFFSET:=
       R_CF_BYT:=
       R_CF_BIT:=
       DONE     :=
       ERROR    :=
       STATUS   :=

      U     "DI_P_SND".DONE
      O     "DI_P_SND".ERROR
      R     "DI_P_SND".REQ              //end of send request
```

```
Network 2: reset PLC restart flag
```

```
      SET
      R     "PLCrestart"
```

S7 code example, English instruction set

OB100

```
Network 1: PLC restart flag

      SET
      S     "PLCrestart"                  //PLC restart flag


Network 2: setup MODBUS SLAVE

      L     17
      T     "drvSlaveDB".setup.SlaveNr  //Modbus slave ID number

      L     101
      T     "drvSlaveDB".setup.RXmailboxDBnr    //number of RX mailbox datablock
      L     102
      T     "drvSlaveDB".setup.TXmailboxDBnr    //number of TX mailbox datablock
      L     110
      T     "drvSlaveDB".setup.Coils_DBnr    //number of the DB containing coils
      L     111
      T     "drvSlaveDB".setup.InputRegisters_DBnr    //number of the input regs DB
      L     112
      T     "drvSlaveDB".setup.HoldingRegisters_DBnr  //number of the holding regs DB
      L     'D'
      T     "drvSlaveDB".setup.CoilsRedirection   //area where coils are redirected to
      SET
      S     "drvSlaveDB".setup.EnableCoils        //enable memory areas
      S     "drvSlaveDB".setup.EnableInputRegisters
      S     "drvSlaveDB".setup.EnableHoldRegisters
      S     "drvSlaveDB".cmd.init          //request to initialize the software driver
```

OB1

```
Network 1: call CP and MODBUS blocks

  CALL  "P_RCV_RK_OLD" , "DI_P_RCV"
        EN_R    :=TRUE
        R       :="PLCrestart"
        LADDR   :=256                   //CP I/O address
        DB_NO   :=101                   //CP incoming data DB
        DBB_NO  :=0                     //incoming data byte offset
        L_TYP   :=
        L_NO    :=
        L_OFFSET:=
        L_CF_BYT:=
        L_CF_BIT:=
        NDR     :=
        ERROR   :=
        LEN     :=
        STATUS  :=

      A     "DI_P_RCV".NDR              //<-- "new data ready" from CP
      R     "DI_P_RCV".NDR
      JCN   nRx
      L     "DI_P_RCV".LEN              //<-- "amount of bytes received" from CP
      T     "drvSlaveDB".cmd.RxAmount
      SET
      S     "drvSlaveDB".cmd.start      //<-- command to start MODBUS data processing
nRx:  NOP   0

      CALL  "drv_slave" , "drvSlaveDB"  //call MODBUS SLAVE driver

      A     "drvSlaveDB".output.SendRequest    //<-- output flag from MODBUS driver
      R     "drvSlaveDB".output.SendRequest
      JCN   nTx
      L     "drvSlaveDB".output.DataAmount
      T     "DI_P_SND".LEN              //<-- "amount of bytes to send" to the CP
      SET
      S     "DI_P_SND".REQ              //<-- set "transmit request" to the CP
nTx:  NOP   0


      CALL  "P_SND_RK_OLD" , "DI_P_SND"
       SF       :=
       REQ      :=
       R        :="PLCrestart"
       LADDR    :=256                  //CP I/O address
       DB_NO    :=102                  //CP outgoing data DB
       DBB_NO   :=0                    //outgoing data offset
       LEN      :=
       R_CPU_NO:=
       R_TYP    :=
       R_NO     :=
       R_OFFSET:=
       R_CF_BYT:=
       R_CF_BIT:=
       DONE     :=
       ERROR    :=
       STATUS   :=

      A     "DI_P_SND".DONE
      O     "DI_P_SND".ERROR
      R     "DI_P_SND".REQ              //end of send request


Network 2: reset PLC restart flag

      SET
      R     "PLCrestart"
```

Byte order issues

Endian byte order can be corrected at PLC side by using TAW and TAD instructions in Simatic PLC program to adjust the byte order according to the different format to be read from or written to the partner.

16-bit values

If not correctly handled by the partner, the byte order can be adapted in the S7 PLC by reversing the low and high order bytes. Use the TAW instruction.

```
L     DB50.DBW   30              //value in Big Endian byte order (e.g. 1234 hex)
TAW                              //swap low and high order bytes
T     "MBUS InputRegisters_DB".Reg[23]    //Little Endian byte order (e.g. 3412 hex)
```

32-bit values

While not specified in the original Modbus protocol, 32-bit values (e.g. float variables) can be transmitted or received by using two contiguous 16-bit registers.
If not correctly handled by the partner Modbus, the byte order can be adapted in the S7 PLC by reversing the bytes order. Use the TAD instruction.

```
L     DB52.DBD   12              //value in Big Endian byte order (e.g. 12345678 hex)
TAD                              //reverse the order of the 4 bytes
T     DB111.DBD   62             //send Little Endian byte order (e.g. 78563412 hex)
```

Other cases

Modbus is a 16-bit registers oriented protocol. Larger types (e.g. 32-bit types) can still be transported but, in the case, must be correctly reassembled by the communication partners.

As an example, a DCS and a PLC must exchange 32-bit float values.
To verify the behaviour of each partner, do as follows:

- Have the DCS send the value DW#16#12345678 and check what is actually being received at PLC side.
- Then have the PLC send the value DW#16#12345678 and check what is actually being received at DCS side.
- Upon results, take measures at either PLC side, DCS side or both.

Troubleshooting

xCP driver error codes

While the xCP Function Block manages exception messages to be sent as Modbus telegrams, it also provides an error word containing detailed diagnostic information. The error word is cleared internally at the beginning of each telegram processing.

Example:

```
L     "drvSlaveDB".output.ErrorCode    //error word from xCP FB
L     0
<>I
=     M 50.0                           //detect protocol error
```

xCP error code (decimal)	Description	MODBUS exception telegram
1	received message from CP is too short, expected at least 6 characters	None
2	CRC error in incoming message	None
3	function code not implemented in the S7 driver	Exception 01 "illegal function"
4	Function FC01: incorrect coils amount. Amount must be greater than zero and not more than 16.	Exception 02 “illegal data address”
5	Function FC01: no redirection area selected	Exception 02 “illegal data address”
6	Function FC01: coils address exceeds memory flags system area in PLC	Exception 02 “illegal data address”
7	Function FC01: coils address exceeds coils DB length	Exception 02 “illegal data address”
8	Function FC01: coils address exceeds holding registers DB length	Exception 02 “illegal data address”
9	Function FC03: incorrect registers amount	Exception 02 “illegal data address”
10	Function FC03: error from SFB20 BLKMOV	Exception 02 “illegal data address”
11	Function FC04: incorrect registers amount	Exception 02 “illegal data address”
12	Function FC04: error from SFB20 BLKMOV	Exception 02 “illegal data address”
13	Function FC05: incorrect redirection area	Exception 02 “illegal data address”
14	Function FC05: incorrect coil true/false value	Exception 01 "illegal function"
15	Function FC05: coil address exceeds memory flags system area in PLC	Exception 02 “illegal data address”
16	Function FC05: coil address exceeds assigned Coils DB area in PLC	Exception 02 “illegal data address”
17	Function FC05: coil address exceeds assigned holding registers DB area in PLC	Exception 02 “illegal data address”
18	Function FC06: request exceeds holding registers DB area in PLC	Exception 02 “illegal data address”
19	Function 08: subfunction number not implemented	Exception 01 "illegal function"
20	Function FC16: incorrect registers amount	Exception 02 “illegal data address”
21	Function FC16: error from SFB20 BLKMOV	Exception 02 “illegal data address”