RUNMODE xCP MODBUS

LOADABLE BLOCKS FOR SIEMENS S7-300/-400 PLC

MODBUS RTU MASTER

CP-independent driver suitable for any communication processors (CP) with serial interface and any PtP CPUs with on-board serial port.

User manual

Documentation last update: January 15, 2011

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

RUNMODE S7 xCP MODBUS RTU MASTER 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 (e.g. 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.

Implemented functions

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

- FC01 read coils
- FC02 read inputs
- FC03 read holding registers
- FC04 read input registers
- FC05 write single coil
- FC06 write single holding register
- FC16 write multiple holding registers

PLC memory footprint

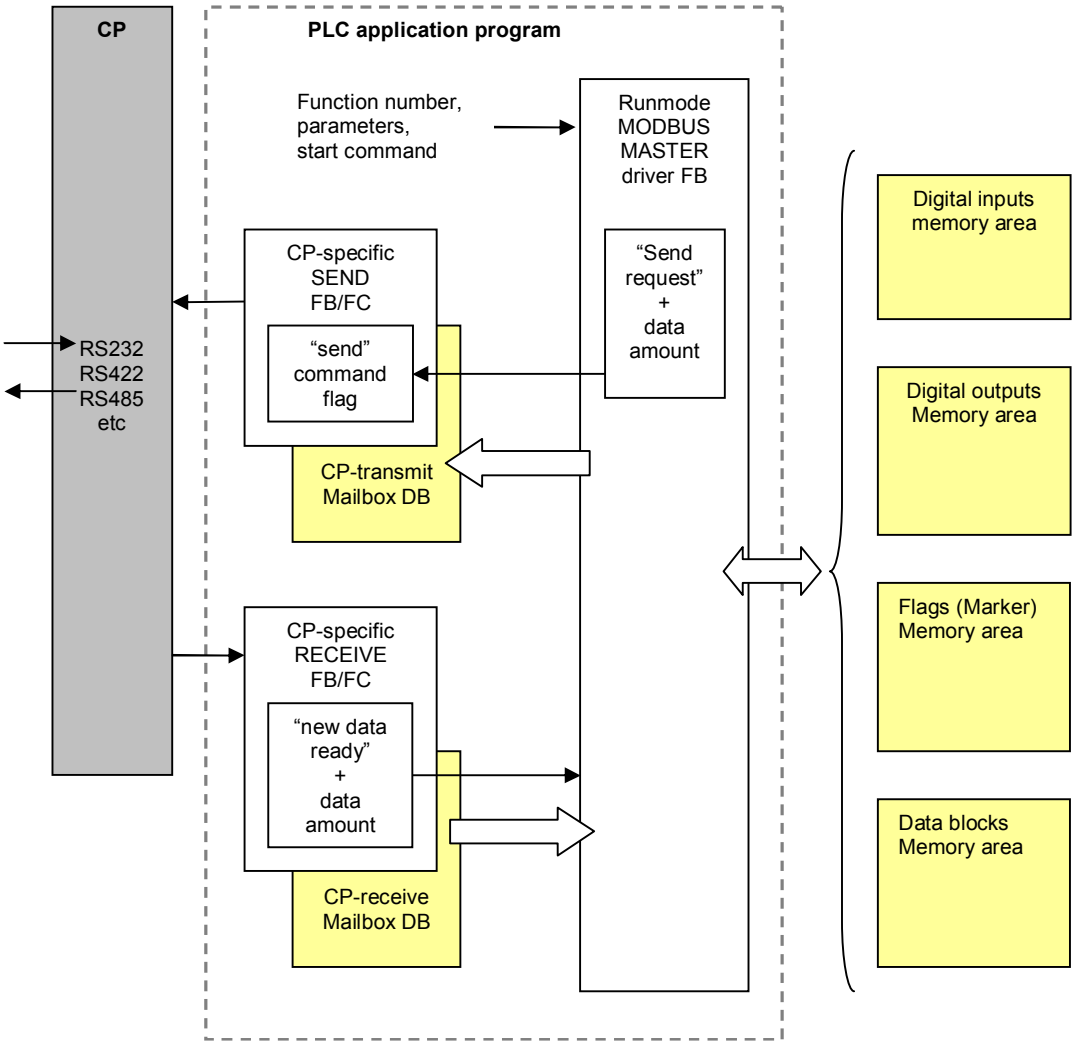
The driver is made of two code blocks, namely FB102 and FC100, than can be renamed. The driver needs also some data space to allocate the send/receive mailboxes plus the instance DB for FB102

Block	description	Local data (bytes)	MC7 - machine code S7 (bytes)	Load memory (bytes)	Work memory (bytes)
FB102	driver	42	2204	2546	2240
FC100	CRC generator	20	198	306	234
DB	FB102 instance DB	- - -	70	254	106
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. The application program fills in the communication job of the MODBUS driver, declaring the slave number, the desired function code and related parameters.
2. The application program sets the “start” flag to initiate the request
3. The MODBUS driver checks continuously the “start” flag. If “true”, the flag is immediately reset while a “busy” flag is set. The driver begins the message processing: it checks for the valid contents of the requested function and takes related actions according to the read or write request. Data to be sent is fetched from the assigned area (digital inputs, digital outputs, flags, data block) and copied to the transmit mailbox DB.
4. The MODBUS driver provides a “send request” flag along with the indication of how many bytes must be sent.
5. The application program must check the state of the “send request” flag and activate the related CP send request. The “send request” flag must be reset by the user program.
6. Once a message has been received from the CP, the application program must forward the amount of received data and reception flag “rx data ready” to the MODBUS driver.
7. The MODBUS driver checks continuously the “rx data ready” flag. If “true”, the flag is immediately reset. The driver begins the message processing: it checks for the valid contents of the incoming data and takes related actions according to the read or write request. Data to be read is then written to the assigned area (digital inputs, digital outputs, flags, data block).


CP
PLC application program
Function number,
parameters,
start command
Runmode
MODBUS
MASTER
driver FB
CP-specific
SEND
FB/FC
"send"
command
flag
"Send
request"
+
data
amount
CP-transmit
Mailbox DB
RS232
RS422
RS485
etc
CP-specific
RECEIVE
FB/FC
"new data
ready"
+
data
amount
CP-receive
Mailbox DB
Digital inputs
memory area
Digital outputs
Memory area
Flags (Marker)
Memory area
Data blocks
Memory area

Functions details

Calling jobs

A job is activated by filling in the related parameters' table and then setting the "start" command flag. Not all parameters are always needed, so refer to the following tables for the list of necessary parameters related to each specific Modbus function.

NOTE : The parameters area is shared among all jobs, do not modify the data while a job is in progress.

Job table	
Slave number	1...255
Function number	1
First element	1...65535
ElementsAmount	1…2000
SingleRegisterValue	W#16#0...W#16#FFFF
SingleBoolValue	0…1
SendDataAreaID	"I", "Q", "M", "D"
SendDBnr	DB number (if area = "D")
SendDataOffset	0...n
ReceiveDataAreaID	"I", "Q", "M", "D"
ReceiveDBnr	DB number (if area = "D")
ReceiveDataOffset	0...n

NOTE: the block accepts both English and German mnemonics for the DataAreaID parameters, either in upper or lower case:

English	**German**
"I", "Q", "M", "D"	"E", "A", "M", "D"
"i", "q", "m", "d"	"e", "a", "m", "d"

The driver outputs a series of status flags of the job:

- output.Busy
- output.FunctionDone
- output.FunctionError

A job ends always with "FunctionDone" or "FunctionError" output flags.

Job parameters

- Slave number: number of the slave to be queried.
- Function number: number of the MODBUS function.
- First element: number of the first partner's element in a row to be addressed, starting from element 1 up to element 65535. An element may be a bit or a 16-bit word register.
- ElementsAmount: number of elements to be addressed, starting from 1.
- SingleRegisterValue: enter here the value for functions where a single register is involved (e.g. function FC06).
- SingleBoolValue: enter here the value for functions where a single bit is involved (e.g. function FC05).
- SendDataAreaID: identification of the memory area to be sent:
  I = input
  Q = output
  M = memory flags
  D = data block
- SendDBnr: number of the data block containing the data to be sent, necessary only if SendDataAreaID has been set to "D"
- SendDataOffset: offset of beginning of the memory area to be sent, based on address 0 as first element (e.g. offset = zero for flag M0.0, offset = zero for DBW 0).
- ReceiveDataAreaID: identification of the memory area where received data will be allocated:
  I = input
  Q = output
  M = memory flags
  D = data block
- ReceiveDBnr: number of the data block which will contain the received data, necessary only if ReceiveDataAreaID has been set to "D"
- ReceiveDataOffset: offset of beginning of allocation in the selected memory area, based on address 0 as first element (e.g. offset = zero for flag M0.0, offset = zero for DBW 0).

Job example

Slave 5, read 4 memory flags starting from flag 8. Result will be stored in DB20 starting from byte 12 (DB12.DBX0.0 onward).
The function 01 does not evaluate SendData parameters.

Job table for FC01 (read coils)	
Slave number	5
Function number	1
First element	8
ElementsAmount	4
SingleRegisterValue	
SingleBoolValue	
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	"D"
ReceiveDBnr	20
ReceiveDataOffset	12

Job code

```
L     5                                 // slave nr = 5
T     "drvMasterDB".cmd.job.SlaveNr

L     1                                 //function FC01
T     "drvMasterDB".cmd.job.FunctionNr

L     11                                //read coils from flag number 11 upward
T     "drvMasterDB".cmd.job.FirstElement

L     5                                 //amount of coils to be read
T     "drvMasterDB".cmd.job.ElementsAmount

L     'D'                               //copy received data to datablock area
T     "drvMasterDB".cmd.job.ReceiveDataAreaID

L     12                                //number of destination datablock
T     "drvMasterDB".cmd.job.ReceiveDataDBnr

L     20                                //starting from byte 20
“drvMasterDB".cmd.job.ReceiveDataOffset

SET                                     //activate the job
S     "drvMasterDB".cmd.job.Start
```

Function 01 : read coils

Read discrete coils: reads a number of coils from the slave memory and store them into the selected receive area.

NOTE: The result is always byte-sized.
If the returned coils quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros.

Job table for FC01 (read coils)	
Slave number	1...255
Function number	1
First element	1...65535
ElementsAmount	1…2000
SingleRegisterValue	
SingleBoolValue	
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	"I", "Q", "M", "D"
ReceiveDBnr	DB number (if area = "D")
ReceiveDataOffset	0...n

Function 02 : read discrete inputs

Read discrete inputs: reads digital inputs from the slave memory, commonly defined as 1xxxxxx registers, and store them into the selected receive area.

NOTE: The result is always byte-sized.
If the returned coils quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros.

Job table for FC02 (read inputs)	
Slave number	1..255
Function number	2
First element	1...65535
ElementsAmount	1...2000
SingleRegisterValue	
SingleBoolValue	
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	"I", "Q", "M", "D"
ReceiveDBnr	DB number (if area = "D")
ReceiveDataOffset	0...n

Function 03 : read holding registers

Read holding registers: reads read/write registers from the slave memory, commonly defined as 4xxxxxx registers, and store them into the selected receive area.

Job table for FC03 (read holding registers)	
Slave number	1...255
Function number	3
First element	1...65535
ElementsAmount	1...125
SingleRegisterValue	
SingleBoolValue	
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	"I", "Q", "M", "D"
ReceiveDBnr	DB number (if area = "D")
ReceiveDataOffset	0...n

Function 04 : read input registers

Read input registers: reads read-only registers in the PLC memory, commonly defined as 3xxxxxx registers, and store them into the selected receive area.

Job table for FC04 (read input registers)	
Slave number	1...255
Function number	4
First element	1...65535
ElementsAmount	1...125
SingleRegisterValue	
SingleBoolValue	
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	"I", "Q", "M", "D"
ReceiveDBnr	DB number (if area = "D")
ReceiveDataOffset	0...n

Function 05 : write single coil

The function sets a coil value in the slave memory upon the status of the SingleBoolValue parameter.

Job table for FC05 (write single coil)	
Slave number	1...255
Function number	5
First element	1...65535
ElementsAmount	
SingleRegisterValue	
SingleBoolValue	0…1
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	
ReceiveDBnr	
ReceiveDataOffset	

Function 06 : write single register

The function sets a register value in the slave memory upon the value of the SingleRegisterValue parameter.

Job table for FC06 (write single register)	
Slave number	1...255
Function number	6
First element	1...65535
ElementsAmount	
SingleRegisterValue	W#16#0...W#16#FFFF
SingleBoolValue	
SendDataAreaID	
SendDBnr	
SendDataOffset	
ReceiveDataAreaID	
ReceiveDBnr	
ReceiveDataOffset	

Function 16 : write multiple registers

The function writes a series of register to the slave memory, fetching the data from the assigned SendDataArea of the PLC.

Job table for FC06 (write single register)	
Slave number	1...255
Function number	16
First element	1...65535
ElementsAmount	1...125
SingleRegisterValue	
SingleBoolValue	
SendDataAreaID	"I", "Q", "M", "D"
SendDBnr	DB number (if area = "D")
SendDataOffset	0...n
ReceiveDataAreaID	
ReceiveDBnr	
ReceiveDataOffset	

Integrate the driver in your S7 program

Resources

The driver needs the flowing PLC resources:

- 1 FB block
- 1 FC block
- 1 instance data block

Your own S7 project must also provide

- 1 RX mailbox DB (your CP receive area).
- 1 TX mailbox DB (your CP transmit area).

Received data is then allocated into existing PLC memory areas (input, output, memory flags, data blocks) according to the parameterization of the related jobs.

AWL source code

The driver is provided as ready-to-use FB100 and FC100 blocks.
In case the numbering of the blocks does not fit your PLC project, a source code (AWL) text file is provided as separate file.
Follow the instructions to compile the driver with a different FB/FC number:

1. Open your project with Simatic Manager and select the “sources” folder.
2. Open the “Modbus Master AWL source” folder in the xCP project
3. Drag&drop (or copy/paste) the “xCP Modbus Master” source code to your project
4. Open the source file you just copied to your project and replace the FB_NUMBER and FC_NUMBER text (you may use the “search and replace” function) with the block numbers of choice (e.g. FB40 and FC18).
5. Compile the source: two new blocks will be created in the “blocks” folder of your project, according to the numbers you set.
6. All done

Protecting the blocks

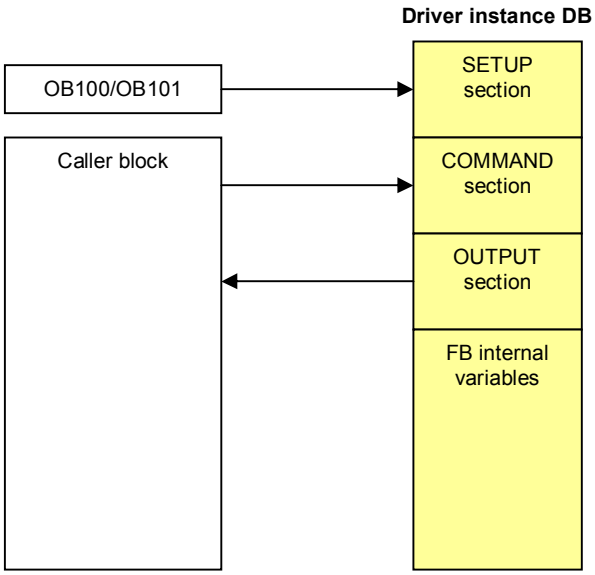
If you like to protect the blocks from being casually accessed or modified, before compiling the source code you may remove comments from the KNOW_HOW_PROTECT keyword listed at the beginning of each block.

NOTE: Be aware that the KNOW_HOW_PROTECT is a rather weak protection and does not guarantee that the code will be not accessible by others.

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.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.SlaveTimeout	0 mS to about 24 days	Slave response timeout. Expressed in S7 TIME format (up to T#+24d20h31m23s647ms)

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 Start	False / True	Start of Modbus query telegram creation. The flag is set by the application program once a job has been parameterized. It is automatically reset by the driver.
DBxx.cmd.job.SlaveNr	1..255	number of the slave to be queried.
DBxx.cmd.job.FunctionNr	1, 2, 3, 4, 5, 6, 16	number of the MODBUS function.
DBxx.cmd.job.FirstElement	1...65535	number of the first partner's element in a row to be addressed, starting from element 1 up to element 65535. An element may be a bit or a 16-bit word register.
DBxx.cmd.job.ElementsAmount	1…2000	number of elements to be addressed, starting from 1.
DBxx.cmd.job.SingleRegisterValue	W#16#0... W#16#FFFF	Register value of functions where a single register is involved (e.g. function FC06).
DBxx.cmd.job.SingleBoolValue	False / True	Bit value of functions where a single bit is involved (e.g. function FC05).
DBxx.cmd.job.SendDataAreaID	"I", "Q", "M", "D"	identification of the memory area to be sent: I = input Q = output M = memory flags D = data block
DBxx.cmd.job.SendDataDBnr	DB number (if area = "D")	number of the data block containing the data to be sent, necessary only if SendDataAreaID has been set to "D"
DBxx.cmd.job.SendDataOffset	0...n	offset of beginning of the memory area to be sent, based on address 0 as first element (e.g. offset = zero for flag M0.0, offset = zero for DBW 0).
DBxx.cmd.job.ReceiveDataAreaID	"I", "Q", "M", "D"	identification of the memory area where received data will be allocated: I = input Q = output M = memory flags D = data block
DBxx.cmd.job.ReceiveDataDBnr	DB number (if area = "D")	number of the data block which will contain the received data, necessary only if ReceiveDataAreaID has been set to "D".
DBxx.cmd.job.ReceiveDataOffset	0...n	offset of beginning of allocation in the selected memory area, based on address 0 as first element (e.g. offset = zero for flag M0.0, offset = zero for DBW 0).
DBxx.cmd.RxDataReady	False / True	Start of Modbus response telegram interpretation. The flag is set by the application program as soon as a new data is received by the Communication processor (e.g. CP340) and it is automatically reset 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 available in the RxMailbox data block.

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.Busy	False / True	The driver is waiting for a response from the slave. Do not modify the current job data while “busy” is TRUE.
DBxx.output.FunctionDone	False / True	Modbus function successfully terminated. The flag is automatically reset every time a “start” command is issued.
DBxx.output.FunctionError	False / True	Modbus function terminated with error. Check the ErrorCode status word. The flag is automatically reset every time a “start” command is issued.
DBxx.output.ExceptionCode	0 .. FFFF hex	It contains the exception code exactly as transmitted by the slave. See the exceptions table in the “troubleshooting” section. The information is available until a new “start” request is received by the driver.
DBxx.output.ErrorCode	0..65535	It contains the code of detailed protocol errors trapped by the driver. See the errors table in the “troubleshooting” section. The information is available until a new “start” request is received by the driver.

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

```
L     100
T     "drvSMasterDB".setup.RXmailboxDBnr    //number of RX mailbox datablock
                                              (incoming data from CP)
L     101
T     "drvMasterDB".setup.TXmailboxDBnr    //number of TX mailbox datablock
                                              (outgoing data to CP)
L     T#2S
T     "drvMasterDB".setup.SlaveTimeout    //slave reply timeout

SET
S     "drvMasterDB".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. assign the job's data and set the "start" flag
3. check for NDR (New data ready) from the CP and forward the received data summary to the xCP MODBUS driver.
4. call xCP MODBUS driver.
5. wait for output data from the driver and forward outgoing data summary to your CP send block.
6. call your CP send block.

Example using CP341 ASCII:

1. call your CP receive block

```
CALL  "P_RCV_RK_OLD" , "DI_P_RCV" // call serial CP "receive" FB
EN_R    :=TRUE
R       :="PLCrestart"
LADDR   :=256
DB_NO   :=100
DBB_NO  :=0
L_TYP   :=
L_NO    :=
L_OFFSET:=
L_CF_BYT:=
L_CF_BIT:=
NDR     :=
ERROR   :=
LEN     :=
STATUS  :=
```

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

```
U     "DI_P_RCV".NDR             //forward notification of received data from CP
=     "drvMasterDB".cmd.RxDataReady
L     "DI_P_RCV".LEN
T     "drvMasterDB".cmd.RxAmount
```

3. set the job “start” flag (may be cyclic or on event)

```
UN    "drvMasterDB".output.Busy
S     "drvMasterDB".cmd.job.Start //<-- issue the START command cyclically
```

4. call the xCP MODBUS driver

```
CALL  "drvModbusMaster" , "drvMasterDB"    // call MODBUS MASTER driver
```

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

```
U     "drvMasterDB".output.SendRequest    //issue transmit request to serial CP
=     "DI_P_SND".REQ
L     "drvMasterDB".output.DataAmount
T     "DI_P_SND".LEN
```

6. call your CP send block

```
CALL  "P_SND_RK_OLD" , "DI_P_SND" //call serial CP "transmit" FB
SF      :=
REQ     :=
R       :="PLCrestart"
LADDR   :=256
DB_NO   :=101
DBB_NO  :=0
LEN     :=
R_CPU_NO:=
R_TYP   :=
R_NO    :=
R_OFFSET:=
R_CF_BYT:=
R_CF_BIT:=
DONE    :=
ERROR   :=
STATUS  :=
```

S7 code example, German instruction set

OB100

```
Network 1: PLC restart flag

      SET
      S     "PLCrestart"                  //PLC restart flag


Network 2: setup the MODBUS MASTER driver

      L     100
      T     "drvMasterDB".setup.RXmailboxDBnr    //number of RX mailbox datablock
                                                   (incoming data from CP)
      L     101
      T     "drvMasterDB".setup.TXmailboxDBnr    //number of TX mailbox datablock
                                                   (outgoing data to CP)
      L     T#2S
      T     "drvMasterDB".setup.SlaveTimeout    //slave reply timeout

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

OB1

```
Network 1: job calls example

        CALL  "jobs_example"


Network 2: Modbus master

        CALL  "P_RCV_RK_OLD" , "DI_P_RCV" // call serial CP "receive" FB
        EN_R    :=TRUE
        R       :="PLCrestart"
        LADDR   :=256
        DB_NO   :=100
        DBB_NO  :=0
        L_TYP   :=
        L_NO    :=
        L_OFFSET:=
        L_CF_BYT:=
        L_CF_BIT:=
        NDR     :=
        ERROR   :=
        LEN     :=
        STATUS  :=

       UN    "drvMasterDB".output.Busy
       S     "drvMasterDB".cmd.job.Start //<-- issue the START command cyclically

       U     "DI_P_RCV".NDR             //forward notification of received data from CP
       =     "drvMasterDB".cmd.RxDataReady
       L     "DI_P_RCV".LEN
       T     "drvMasterDB".cmd.RxAmount


       CALL  "drvModbusMaster" , "drvMasterDB"    // call MODBUS MASTER driver


       U     "drvMasterDB".output.SendRequest    //issue transmit request to serial CP
       =     "DI_P_SND".REQ
       L     "drvMasterDB".output.DataAmount
       T     "DI_P_SND".LEN

       CALL  "P_SND_RK_OLD" , "DI_P_SND" //call serial CP "transmit" FB
        SF      :=
        REQ     :=
        R       :="PLCrestart"
        LADDR   :=256
        DB_NO   :=101
        DBB_NO  :=0
        LEN     :=
        R_CPU_NO:=
        R_TYP   :=
        R_NO    :=
        R_OFFSET:=
        R_CF_BYT:=
        R_CF_BIT:=
        DONE    :=
        ERROR   :=
        STATUS  :=


Network 3: 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 the MODBUS MASTER driver

      L     100
      T     "drvMasterDB".setup.RXmailboxDBnr    //number of RX mailbox datablock
                                                   (incoming data from CP)
      L     101
      T     "drvMasterDB".setup.TXmailboxDBnr    //number of TX mailbox datablock
                                                   (outgoing data to CP)
      L     T#2S
      T     "drvMasterDB".setup.SlaveTimeout    //slave reply timeout

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

OB1

```
Network 1: job calls example

        CALL  "jobs_example"


Network 2: Modbus master

        CALL  "P_RCV_RK_OLD" , "DI_P_RCV" // call serial CP "receive" FB
        EN_R    :=TRUE
        R       :="PLCrestart"
        LADDR   :=256
        DB_NO   :=100
        DBB_NO  :=0
        L_TYP   :=
        L_NO    :=
        L_OFFSET:=
        L_CF_BYT:=
        L_CF_BIT:=
        NDR     :=
        ERROR   :=
        LEN     :=
        STATUS  :=

       AN    "drvMasterDB".output.Busy
       S     "drvMasterDB".cmd.job.Start //<-- issue the START command cyclically

       A     "DI_P_RCV".NDR             //forward notification of received data from CP
       =     "drvMasterDB".cmd.RxDataReady
       L     "DI_P_RCV".LEN
       T     "drvMasterDB".cmd.RxAmount


       CALL  "drvModbusMaster" , "drvMasterDB"    // call MODBUS MASTER driver


       A     "drvMasterDB".output.SendRequest    //issue transmit request to serial CP
       =     "DI_P_SND".REQ
       L     "drvMasterDB".output.DataAmount
       T     "DI_P_SND".LEN

       CALL  "P_SND_RK_OLD" , "DI_P_SND" //call serial CP "transmit" FB
        SF      :=
        REQ     :=
        R       :="PLCrestart"
        LADDR   :=256
        DB_NO   :=101
        DBB_NO  :=0
        LEN     :=
        R_CPU_NO:=
        R_TYP   :=
        R_NO    :=
        R_OFFSET:=
        R_CF_BYT:=
        R_CF_BIT:=
        DONE    :=
        ERROR   :=
        STATUS  :=


Network 3: reset PLC restart flag

       SET
       R     "PLCrestart"
```

Troubleshooting

Error word codes

While the xCP Function Block manages exception messages received by Modbus slaves, it also provides an error word with more detailed diagnostic information for the PLC programmer.
The error word is cleared internally at the beginning of each telegram processing.

Example:

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

xCP error code (decimal)	Description
1	function code not implemented in the S7 driver
2	Slave response timeout
3	Slave response contains an unexpected slave number
4	received message from CP is too short, expected at least 6 characters
5	Modbus Exception message received. Check exception codes status word and table.
6	Slave response contains an unexpected function number
7	CRC error in incoming message
8	Function FC1/2/3/4: error from SFB20 BLKMOV
9	Function FC05: incorrect returned written address
10	Function FC05: incorrect returned written value
11	Function FC06: incorrect returned written address
12	Function FC06: incorrect returned written value
13	Function FC16: incorrect returned written address
14	Function FC16: incorrect returned number of written registers

Exception codes

The exceptions are error codes sent by the slave and must comply to the following Modbus standard list:

Code	Name	Meaning
01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
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, a request with offset 96 and length 5 will generate exception 02.
03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.
04	SLAVE DEVICE FAILURE	An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
05	ACKNOWLEDGE	Specialized use in conjunction with programming commands. The server (or slave) has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client (or master). The client (or master) can next issue a Poll Program Complete message to determine if processing is completed.
06	SLAVE DEVICE BUSY	Specialized use in conjunction with programming commands. The server (or slave) is engaged in processing a long–duration program command. The client (or master) should retransmit the message later when the server (or slave) is free.
08	MEMORY PARITY ERROR	Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server (or slave) attempted to read record file, but detected a parity error in the memory. The client (or master) can retry the request, but service may be required on the server (or slave) device.
0A	GATEWAY PATH UNAVAILABLE	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request.
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

Known issues

Problems with CUBLOC PLC

We had an issue reported by a customer while communicating with a CUBLOC plc acting as Modbus slave.

It is not a problem of our xCP driver but rather a bug in the CUBLOC Modbus management, for the CUBLOC, when using function 06, does not reply exactly as per Modbus standard specifications.

The solution is simple: when using function 06, just provide the full CUBLOC memory area address (e.g. register 0x7002) instead of 0x0002 (as you would do in a normal Modbus query).

Other issues

No issues with any other PLC or device are reported.