

ADDI-DATA GmbH Dieselstraße 3 D-77833 OTTERSWEIER





Preliminary version

Technical description

MSX-Box

S7 communication

Edition: 01.01 - 02/2007



Product information

This manual contains the technical installation and important instructions for correct commissioning and usage, as well as production information according to the current status before printing. The content of this manual and the technical product data may be changed without prior notice. ADDI-DATA GmbH reserves the right to make changes to the technical data and the materials included herein.

Warranty and liability

The user is not permitted to make changes to the product beyond the intended use, or to interfere with the product in any other way.

ADDI-DATA shall not be liable for obvious printing and phrasing errors. In addition, ADDI DATA, if legally permissible, shall not be liable for personal injury or damage to materials caused by improper installation and/or commissioning of the board by the user or improper use, for example, if the board is operated despite faulty safety and protection devices, or if notes in the operating instructions regarding transport, storage, installation, commissioning, operation, thresholds, etc. are not taken into consideration. Liability is further excluded if the operator changes the board or the source code files without authorisation and/or if the operator is guilty of not monitoring the permanent operational capability of working parts and this has led to damage.

Copyright

This manual, which is intended for the operator and its staff only, is protected by copyright. Duplication of the information contained in the operating instructions and of any other product information, or disclosure of this information for use by third parties, is not permitted, unless this right has been granted by the product licence issued. Non-compliance with this could lead to civil and criminal proceedings.

ADDI-DATA software product licence

Please read this licence carefully before using the standard software. The customer is only granted the right to use this software if he/she agrees with the conditions of this licence.

The software must only be used to set up the ADDI-DATA boards.

Reproduction of the software is forbidden (except for back-up and for exchange of faulty data carriers). Disassembly, decompilation, decryption and reverse engineering of the software are forbidden. This licence and the software may be transferred to a third party if this party has acquired a board by purchase, has agreed to all the conditions in this licence contract and the original owner does not keep any copies of the software.

Trademarks

- ADDI-DATA is a registered trademark of ADDI-DATA GmbH.
- Turbo Pascal, Delphi, Borland C, Borland C++ are registered trademarks of Borland Insight Company.
- Microsoft C, Visual C++, Windows XP, 98, Windows 2000, Windows 95, Windows NT, EmbeddedNT and MS DOS are registered trademarks of Microsoft Corporation.
- LabVIEW, LabWindows/CVI, DasyLab, Diadem are registered trademarks of National Instruments Corp.
- CompactPCI is a registered trademark of PCI Industrial Computer Manufacturers Group.
- VxWorks is a registered trademark of Wind River Systems Inc.

WARNING

The following risks result from improper implementation and from use of the board contrary to the regulations:

- Personal injury
- Damage to the board, MSX-Box, PC and peripherals
- Pollution of the environment
- Protect yourself, the others and the environment!
- Read carefully the safety precautions (yellow leaflet).

If this leaflet is not with the documentation, please contact us and ask for it.

• Observe the instructions of the manual.

Make sure that you do not forget or skip any step. We are not liable for damages resulting from a wrong use of the board.

• Used symbols:

IMPORTANT!

Designates hints and other useful information.



WARNING!

It designates a possibly dangerous situation. If the instructions are ignored the board, PC and/or peripheral may be destroyed.

1	INTRODUCTION	7
1.1	About this documentation	7
1.2	Requirements	7
1.3	Description	7
2	OPEN MODBUS TCP	9
2.1	Introduction	9
2.2	Type definition	9
2.3	Header description	9
2.4	Class 0: Read multiple registers (FC3)	10
2.4.1	Request	
2.4.2 2.4.3	Response Exception	
2.5	Class 0: Write multiple registers (FC16)	
2.5.1	Request	
2.5.2	Response	
2.5.3	Exception	12
3	MSX-BOX	13
3.1	OPEN MODBUS TCP Slave package	13
3.1 3.2	OPEN MODBUS TCP Slave package Steps to follow	
		14
3.23.33.3.1	Steps to follow Skeleton functions	14 15 15
3.2 3.3 3.3.1 3.3.2	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart	14 15 15
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4	Steps to follow Skeleton functions	14 15 15 15 16
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters	14 15 15 16 17 18
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters	14 15 15 16 17 18
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters	14 15 15 16 16 17 18 18 121
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters Installation	14 15 15 16 16 17 18 18 17 18 121
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4 3.5	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters	14 15 15 16 16 17 18 18 17 18 121
 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4 3.5 3.4 3.5 3.6 3.7 3.7.1 	Steps to follow. Skeleton functions. Introduction. skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPServerStop. skel_MODBUSTCPReadMultipleRegisters. skel_MODBUSTCPWriteMultipleRegisters. Compilation Execution. Installation Boot script Description	14 15 15 15 16 17 18 21 21 21 22 22 22
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4 3.5 3.6 3.7	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters Description Use of the script	14 15 15 16 16 17 18 21 21 21 22 22 23 23
 3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4 3.5 3.4 3.5 3.6 3.7 3.7.1 	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Steps to follow Description Use of the script S7 PROGRAM	14 15 15 16 16 17 18 21 21 21 22 22 22 23 23 24
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4 3.5 3.4 3.5 3.6 3.7 3.7.1 3.7.2	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters Skel_MODBUSTCPWriteMultipleRegisters Description Use of the script	14 15 15 16 16 17 18 21 21 21 22 22 22 23 23 24
3.2 3.3 3.3.1 3.3.2 3.3.3 3.3.4 3.3.5 3.4 3.5 3.4 3.5 3.6 3.7 3.7.1 3.7.2 4	Steps to follow Skeleton functions Introduction skel_MODBUSTCPServerStart skel_MODBUSTCPServerStop skel_MODBUSTCPReadMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters skel_MODBUSTCPWriteMultipleRegisters Steps to follow Description Use of the script S7 PROGRAM	14 15 15 16 16 16 17 18 21 21 22 22 22 23 24 24

4.3.1	OB1	
4.3.2	OB100	
4.3.3	FB100	
4.3.4	FC10	
5	COMMENTS	40
5.1	S7 programs	40

Figures

Fig. 4-1: SIMATIC Manager	
Fig. 4-2: NetPro	25
Fig. 4-3: Properties – TCP connection	
Fig. 4-4: OB1	
Fig. 4-5: OB100	
Fig. 4-6: FB100	30
Fig. 4-7: FC10	32

Tables

Table 2-1: Open Modbus TCP: Header description	9
Table 2-2: Read multiple registers FC3: Request	10
Table 2-3: Read multiple registers FC3: Response	10
Table 2-4: Read multiple registers FC3: Exception	10
Table 2-5: Write multiple registers FC16: Request	11
Table 2-6: Write multiple registers FC16: Response	12
Table 2-7: Read multiple registers FC16: Exception	12

1 INTRODUCTION

1.1 About this documentation

This documentation describes how to control a **MSX-Box** with a Siemens **S7** over Ethernet.

It is based on a sample using the **APCI-1500** (a board with 16 digital inputs and outputs) as hardware to be managed (in the **MSX-Box**) through the **S7**.

The S7 is reading the status of the 16 digital inputs of the APCI-1500.

Hardware:

- o MSX-Box
- o APCI-1500 (16 digital inputs / outputs)
- Siemens S7 + CP343-1 Lean

The Siemens **S7** has to read and write digital inputs from the **APCI-1500** (located in the **MSX-Box**).

1.2 Requirements

Please make sure that the following requirements are fulfilled:

- o Siemens CPU313C-2DP (The PLC device)
- o Siemens CP343-1 Lean (Ethernet module for the PCL)
- FC5 (AG_SEND) and FC6 (AG_RECV) blocks for S7-300 (asynchrony communication function)
- o MSX-Box OPEN MODBUSTCP Slave package
- S7 sample for Step 7

1.3 Description

The Siemens **S7** is used as a master to remote the slave **MSX-Box** by using the OPEN MODBUS TCP protocol. The **MSX-Box** is running an OPEN MODBUS TCP Slave server which is waiting for the master request.

ADDI-DATA provides a **MSX-Box** OPEN MODBUS TCP slave server servicing Class 0 functions.

This class is the minimal set of functions that has to be available on a device providing the OPEN MODBUS TCP protocol.

Class 0 includes a set of 2 functions:

- Read multiple registers
- Write multiple registers

On the **MSX-Box** side, OPEN MODBUS TCP telegrams do not need to be manipulated directly. This is already done by the server. Read and write skeletons functions have to be filled with the code they have to execute once the **S7** asks for reading or writing. On the **S7**, OPEN MODBUS TCP telegrams have to be set (See <u>S7 Program</u>).

2 OPEN MODBUS TCP

2.1 Introduction

The OPEN MODBUS TCP protocol is based on the widely known MODBUS protocol.

OPEN MODBUS TCP is an open protocol and is not manufacturer dependent.

It is mainly used to connect PLC and I/O devices.

The OPEN MODBUS TCP protocol is using the connection oriented TCP protocol in order to ensure security features and simplify the server and client codes. Data are encoded in big-endian (for data larger as bytes, the most significant byte is sent first).

The OPEN MODBUS TCP telegram is composed of two parts, a header and a body.

2.2 Type definition

Please note: 1 x byte = 8-bit 1 x word = 16-bit = 2 x bytes

2.3 Header description

The header is always composed of 6 bytes:

Table 2-1: Open Modbus TCP	: Header description
----------------------------	----------------------

Byte	Signification		Value	Comment
0	Transaction identifier	MSB	0	Copied by server
1	Transaction identifier	LSB	0	
2	Protocol identifier	MSB	0	OPEN MODBUS TCP =
3	Protocol identifier	LSB	0	0
4	Length field	MSB	0	0 Because messages are
5	Length field	LSB	Number of bytes of the body	smaller than 256

MSB: Most significant byte

LSB: Least significant byte

2.4 Class 0: Read multiple registers (FC3)

2.4.1 Request

Table 2-2: Read multiple registers FC3: Request

Byte	Signification		Value	Comment
0	Unit identifier			Index of the slave to be controlled
1	Function code		3	Code of the function to execute
2	Reference number	MSB		First register to be read
3	Reference number	LSB		
4	Word count	MSB	1-125	Number of words to be read from
5	Word count	LSB		the reference register

MSB: Most significant byte LSB: Least significant byte

2.4.2 Response

Table 2-3: Read multiple registers FC3: Response

Byte	Signification	Value	Comment
0	Unit identifier		Index of the slave to be controlled
1	Function code	3	Code of the function to execute
2	Byte count of response		Number of words from the response converted in byte.
3	Register values		Read words

2.4.3 Exception

Table 2-4: Read multiple registers FC3: Exception

Byte	Signification	Value	Comment
0	Unit identifier		Index of the slave to be controlled
1	Function exception code	83	Function code + Exception modifier
2	Exception code	1	1: Illegal function
		or	
		2	2: Illegal data address

Sample:

Read 2 registers at reference 8 slave 5.

Reference	Register value (Hex)	
8	5678	
9	4897	

Request:00 00 00 00 00 00 06 05 03 00 08 00 02Response:00 00 00 00 00 07 05 03 04 56 78 48 97

OPEN MODBUS TCP Header

2.5 Class 0: Write multiple registers (FC16)

2.5.1 Request

Table 2-5: Write multiple registers FC16: Request

Byte	Signification		Value	Comment
0	Unit identifier			Index of the slave to be controlled
1	Function code		16	Code of the function to execute
2	Reference number	MSB		First register to be written
3	Reference number	LSB		
4	Word count	MSB	1-100	Number of words to be written
5	Word count	LSB		from the reference register
6	Byte count			Number of bytes to be written from the reference register
7 	Register values			Words to be written from the reference register

MSB: Most significant byte LSB: Least significant byte

2.5.2 Response

Table 2-6: Write multiple registers FC16: Response

Byte	Signification		Value	Comment
0	Unit identifier			Index of the slave to
1	Function code		16	Code of the function to execute
2	Reference number	MSB		This is the first written register
3	Reference number	LSB		
4	Word count	MSB		The number of written words
5	Word count	LSB		

MSB: Most significant byte LSB: Least significant byte

2.5.3 Exception

Table 2-7: Read multiple registers FC16: Exception

Byte	Signification	Value	Comment
0	Unit identifier		Index of the slave to
1	Function exception code	90	Function code + Exception modifier
2	Exception code	1	1: Illegal function
		or	
		2	2: Illegal data address

Sample:

Write 2 registers at reference 8 slave 5.

Reference	Register value (Hex)
8	5678
9	4897

Request:00 00 00 00 00 0C 05 10 00 08 00 02 00 04 56 78 48 97Response:00 00 00 00 00 06 05 10 00 08 00 02

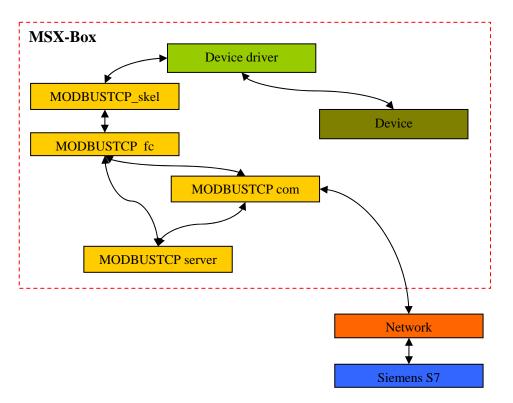
OPEN MODBUS TCP Header

3 MSX-BOX

3.1 OPEN MODBUS TCP Slave package

On the **MSX-Box**, the slave server contains all requested functions for the communication between Siemens **S7**, **MSX-Box** and other peripherals. It is composed of the following parts:

MODBUSTCP_com.c:	Contains all functions to realise the TCP communication.
MODBUSTCP_com.h:	Header of the previous c file.
MODBUSTCP_fc.c:	Contains MODBUS TCP Class 0 functions and exceptions management.
MODBUSTCP_fc.h:	Header of the previous c file.
MODBUSTCP_server.c:	This is where the main functions are located.
MODBUSTCP_skel.c:	Contains read / write skeletons functions which have to be filled.
MODBUSTCP_skel.h:	Header of the previous c file.
MODBUSTCP.h:	Header for MODBUS TCP specific information.



Relation between the different parts:

Parts running in the user level are in yellow (yellow). The device driver kernel level is in green (green).

The code is written so that only the skeleton parts of the slave have to be filled.

3.2 Steps to follow

In order to realise a slave server, the following steps have to be done:

- 1. Fill the hook functions located in MODBUSTCP_skel.c:
 - o skel_MODBUSTCPServerStart
 - o skel_MODBUSTCPServerStop
 - o skel_MODBUSTCPReadMultipleRegisters
 - o skel_MODBUSTCPWriteMultipleRegisters
- 2. Compile the slave server.

3.3 Skeleton functions

3.3.1 Introduction

Skeleton functions are like hook functions which are called when the defined task is done. You have to fill the body of these hook functions as you require.

Four functions are available in this slave version:

- o skel_MODBUSTCPServerStart
- o skel_MODBUSTCPServerStop
- o skel_MODBUSTCPReadMultipleRegisters
- o skel_MODBUSTCPWriteMultipleRegisters

3.3.2 skel_MODBUSTCPServerStart

Prototype:

int skel_MODBUSTCPServerStart (void)

When it is called:

- Once the slave application is started

- Initialization and request waiting.

Has to be used for:

Allows you to make your own initialization before the slave starts. This is a good place to open and/or initialize the hardware that has to be managed by the S7.

Parameters:

No parameter.

Return value:

If you return a value different from 0, the server does not start. E.g.: If the hardware to use is not available, a value different of 0 can be returned. The slave server does not start.

Sample:

In this sample the used hardware is an ADDI-DATA **APCI-1500** board. The apci1500_find_cards functions searches for an **APCI-1500** in the **MSX-Box** and returns the number of boards found.

If no board was found, it returns 1 (error) and the slave server will not go on running.

```
int skel_MODBUSTCPServerStart (void)
{
    /* Search all APCI-1500 boards and return the
number of boards */
    if ((nbr = apci1500_find_cards(&fd)) < 1)
    {
        return 1;
    }
    return 0;
}</pre>
```

3.3.3 skel_MODBUSTCPServerStop

Prototype:

int skel_MODBUSTCPServerStop (void)

When it is called:

Just before the slave server quits (because of error or user has quit it).

Has to be used for:

Allows to release before the slave is quitting. This is a good place to close and/or release the hardware that has to be managed by the **S7**.

Parameters:

No parameter.

Return value:

Not used.

Sample:

In this sample the used hardware is an ADDI-DATA **APCI-1500** board. Set all digital outputs of the APCI-1500 to 0 before quitting the slave server.

3.3.4 skel_MODBUSTCPReadMultipleRegisters

Prototype:

```
uint16_t skel_MODBUSTCPReadMultipleRegisters(uint16_t reference, uint16_t
length, uint16_t *value)
```

When it is called:

Once the slave received a valid (valid = the telegram is tested from the slave server) FC3 request telegram. FC3 is reading multiple registers.

Has to be used for:

Read something or do an action.

Parameters:

Input:

Reference Length	Index of the first register to be read. Number of words to read from the reference register.
Output:	
Value	This is a word array in which you have to return read words. The array size is "length" size!

Return value:

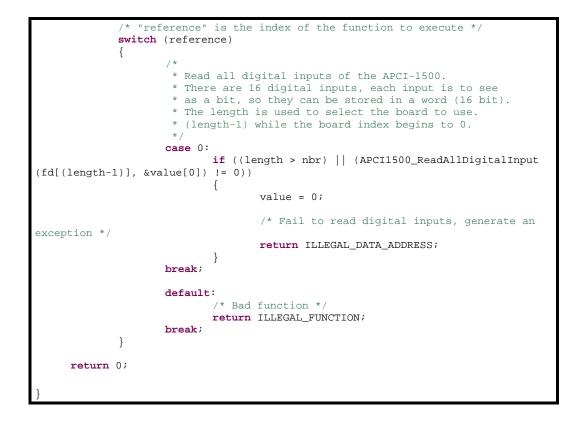
- 0: No error
- 1: ILLEGAL_FUNCTION
- 2: ILLEGAL_DATA_ADDRESS
- 3: ILLEGAL_DATA_VALUE (Not specified for this FC in the MODBUSTCP specification, but can be useful in some case)

Sample:

In this sample, the reference is used as a function index. Here reference = 0 calls the APCI1500_ReadAllDigitalInput function. Length is used as an index to specify the board to use; it is the minor number of the board. Value will contains the status of the 16 digital inputs of the APCI-1500.

Remark: Data in the value array do not have to be converted in big-endian for the communication. The slave server does this automatically for you.

uint16_t skel_MODBUSTCPReadMultipleRegisters(uint16_t reference, uint16_t length, uint16_t *value)



3.3.5 skel_MODBUSTCPWriteMultipleRegisters

Prototype:

```
uint16_t skel_MODBUSTCPWriteMultipleRegisters(uint16_t reference, uint16_t
length, uint16_t *value)
```

When it is called:

Once the slave receives a valid (valid = the telegram is tested from the slave server) FC16 request telegram. FC16 is writing multiple registers.

Has to be used for:

Write something or do an action.

Parameters:

Input:

reference	Index of the first register to be written.
length	Number of words to write from the reference register.
value	This is a word array in which you will find words to write
The array size	is "length" size!

Return value:

- 0: No error
- 1: ILLEGAL_FUNCTION

- 2: ILLEGAL_DATA_ADDRESS
- 3: ILLEGAL_DATA_VALUE (Not specified for this FC in the MODBUSTCP specification, but can be useful in some case)

Sample:

In this sample, the reference is used as a function index. The reference = 0 calls the APCI1500_WriteAllDigitalInput function. Length is used as an index to specify the board to use. It is the minor number of the board. Value contains the status value to write on the 16 digital outputs of the APCI-1500.

Comment: Data in the value array do not have to be converted from big-endian due to the communication. The slave server does it automatically for you.

```
uint16_t skel_MODBUSTCPWriteMultipleRegisters(uint16_t
reference, uint16_t length, uint16_t *value)
ł
          /* "reference" is the index of the function
to execute */
          switch (reference)
          {
               /*
                * Write all digital outputs of the
APCI-1500.
                * There are 16 digital outputs, each
output is to see
                * as a bit, so they can be stored in a
word (16 bit).
                * The length is used to select the
board to use.
                * (length-1) while the board index
begins to 0.
                * /
               case 0:
                    if ((length > nbr) ||
(APCI1500_WriteAllDigitalOutput (fd[(length-1)],
value[0]) != 0))
                    {
                         value = 0;
                         /* Fail to write digital
outputs, generate an exception */
                         return ILLEGAL_DATA_ADDRESS;
                    }
               break;
               default:
                    /* Bad function */
                    return ILLEGAL_FUNCTION;
               break;
          }
     return 0;
```

3.4 Compilation

Once skeletons functions are ready to work, just use the Makefile to compile the complete slave server.

In the slave server directory type "make". The resulting executable is called MODBUSTCP_server.exe (compiled under Cygwin) or MODBUSTCP_server (compiled under Linux).

Note:

For more information about compiling with the **MSX-Box** see "Introduction to C Programming for the MSX-Box.pdf".

3.5 Execution

Transfer the MODBUSTCP_server.exe on the **MSX-Box** (by e.g.: ftp) and use Telnet or the serial connection to open a console on the **MSX-Box**.

The file has to be executable:

[root@MSXBOX:/tmp]# chmod +x MODBUSTCP_server.exe

Usage: %s <slaveAddress> <deamon>

Parameters:

slaveAddress: This is the Unit ID of slave on which the slave server is running.

deamon : 0 =Start a non deamon, 1 = start as deamon. Deamon means that the slave server is running in background and the console on which it is started is free for key inputs.

Sample:

To start it as daemon with Unit ID 2:

[root@MSXBOX:/tmp]# ./MODBUSTCP_server.exe 2 1

To stop it:

[root@MSXBOX:/tmp]# kill MODBUSTCP_server.exe

This daemon logs a big part of the actions that it does. The log is available under /var/log/message:

Feb 8 14:16:15 (none) daemon.warn MODBUS_slave: ...

Note:

For more information about ftp and telnet connections see **QuickInstallation_e.pdf**.

3.6 Installation

If you want to start the server with the boot script:

- the apci1500.0 driver has to be located under: /lib/modules/2.4.xx-x/addidata/
- the MODBUSTCP_server.exe server has to be located under: /home/MODBUSTCP/

You can modify the boot script for other locations.

3.7 Boot script

It is possible to load automatically the slave server by using a script like the following. This sample script loads in first the driver of the **APCI-1500** board.

#!/bin/sh

```
# Starting / Stopping...
#
  (C) ADDI-DATA GmbH 2007
# Module to load
modulesload="apci1500"
# auto build unload string from load string
for i in $modulesload ; do
   modulesunload="$i $modulesunload"
done
# Check presence of all kernel module files listed in "modulesload"
for i in $modulesload ; do
      module="/lib/modules/`uname -r`/addidata/$i.o"
      if [ ! -f $module ] ; then
              echo "$0: $module does not exist!"
              exit 11
      fi
done
start() {
       # Load the APCI-1500 driver
      echo -n "Starting MODBUSTCP server: "
      for i in $modulesload ; do
          modprobe $i
      done
      # Start the MODBUSTCP server as a deamon with slave ID 0
       /home/MODBUSTCP/./MODBUSTCP_server.exe 0 1
      echo "done."
stop()
      {
       # Unload the APCI-1500 driver
      echo -n "Stopping FireWire: "
      for i in $modulesunload ; do
          modprobe -r $i
      done
      # Stop the MODBUSTCP server
      killall MODBUSTCP_server.exe
      echo "done."
restart() {
       # Restart
      Stopp
```

```
start
case "$1" in
   start)
      start
      ;;
    stop)
      stop
      ;;
   restart [reload)
      restart
      ;;
    *)
      echo $"Usage: $0 {start|stop|restart}"
      exit 1
esac
exit $?
```

3.7.1 Description

When the **MSX-Box** is booting, this script is automatically called with the "start" argument (because the link name is SXX like Start).

The "start" function

- o loads the apci1500 driver
- executes the MODBUSTCP_server as a deamon (deamon = is running in background)

To stop manually

[root@MSXBOX:/]# MODBUSTCP stop

To start manually

[root@MSXBOX:/]# MODBUSTCP start

3.7.2 Use of the script

The script e.g.: MODBUSTCP has to be copied in /etc/init.scripts (by using ftp and a consol).

A symbolic link has to be created under /etc/init.d/:

[root@MSXBOX:/tmp]# ln -s /etc/init.scripts/MODBUSTCP /etc/init.d/S98MODBUSTCP

The script will be called and executed at each start of the MSX-Box.

Note:

For more information about ftp and telnet connections see **QuickInstallation_e.pdf**.

4 S7 PROGRAM

4.1 Introduction

The software package contains a sample to read and write values. The documentation describes the sample to read values.

4.2 Configuration

Samples are written for the S7-300 and CP343-1 for the Ethernet communication.

Open the SIMATIC Manager, open the mTCPRead Project.

Fig. 4-1: SIMATIC Manager

🖉 SIMATIC Manager - [mTCPRead D:\Siemens\Step7\s7proj\mTCPRead]							
Eile Edit Insert PLC View Options Window Help							
		< No Filter >	• <u>•</u> 20				
Image: Station SIMATIC 300 Image: Station SIMATIC 300	I Station SIMATIC 300	₩PI(1)	œœ Ethernet(1)				
Press F1 to get Help.		PC Adapter	(MPI)				

• Double-click on Ethernet.

🔀 NetPro - [mTCPRead (Network) D:\Siemens\Step7\s7proj\mTCPRe	ad]			
Retwork Edit Insert PLC Yew Options Window Help			-	8 X
.MPI(1)	1	^		: D X
MPI			End	ntal
			Selection of the network	
			BOFIBUS DP	
Station SIMATIC 300			# PROFIBUS-PA	
IIII CPU CP 340-1 CP 340-1 CP CP S40-1 S40-1			B PROFINET ID	
			Descriptions Description	
3 2				
Ethernet(1)				
Industrial Ethernet				
	2	-		
	Tune	~		
0001 A050 Liaison TCP1	TCP con	-		
				_
		~	PROFIBUS-DP slaves for SIMATIC S7, M7, and C7 (distributed rack)	₹ś
• • • • • • • • • • • • • • • • • • •	Σ		The second of the second	
PC Adapter(MPI) X 4	51 Y 154	11		

Fig. 4-2: NetPro

Click the CPU313C-2DP. In the table, double-click on the marked line (here in black).

Content: The Ethernet connection is a **TCP connection**.

Properties - TCP connection	
Options Overview General Information	Status Information
Local Endpoint	Block Parameters
ID (hex): 0001 A050 ▼ Name: Liaison TCP1 Via <u>C</u> P: CP 343-1 Lean - (R0/S4)	1-ID W#16#0100-LADDR
Route ✓ Active connection establishment ✓ Use ETP protocol	
ОК	CancelHelp

Fig. 4-3: Properties – TCP connection

Active connection establishment has to be selected so that the **S7** does the connection on the slave.

You can see right over the **Block Parameters** that has to be used with the FC5 and FC6 block functions.

Click on Addresses.

Properties - TC	P connection	×					
Ports from 1025	Overview eral Information through 65535 are avai s, refer to online help)	Status Information Addresses lable.					
<u>I</u> P (dec): <u>P</u> ORT (dec):	Local 172.16.3.152 502	Remote 172.16.3.215 502					
OK Cancel Help							

IP and PORT Local are parameters of the **S7**. 502 is the right port number for OPEN MODBUS TCP.

In **Remote**, set the IP address of the **MSX-Box** and the port. Note that the port is always 502, and the IP address has to be compatible with the **S7** IP address.

4.3 Program description

The program is located under the CPU and Blocks.

SIMATIC Manager - [mTCPRead D:\Siemens\Step7\s7proj\mTCPRead]						
Eile Edit Insert PLC View Options Window Help						
D 🛩 🎛 🚿 👗 🖻 🖻	💼 🔍 💁		No Filter >	- <u>7</u>		
TCPRead Torread Station SIMATIC 300 CPU313C-2DP(1) For Program S7(1) For Sources Sources Torread CP 343-1 Lean	Systemdaten FC3 DB101	⊕ 0B1 ₽ FC5 ⊕ 0B102	⊕ 08100 ₽ FC6 ₽ VAT_1	₽ FB100 ₽ FC10 ₽ SFC58	₽ FC2	
Press F1 to get Help.			PC Adap	oter(MPI)		

4.3.1 OB1

What is OB1: OB1 is executed periodically. Once finished it starts again. OB1 is started after the start-up has been completed.

Task: OB1 call the FB100. FB100 (Function Block 100) contains other function calls. M1.0 is here as a flag to test if connection has been initialized once.

Fig. 4-4: OB1

OB1 : Title:				
Comment:				
Network 1: Title:				
Comment:				
DB100 FB100 EN ENO M1.0 - INIT_COM				

Task: When FB100 returns, it resets M1.0, connection is already initialized! It has not to be initialised for the next execution of OB1.

Network 2 : Res	et initial	communication	start
Comment:			
M1.0			M1.0
<u>├</u> ──┤			(R)

4.3.2 OB100

What is OB100: It is called when the CPU is restarted (Warm restart).

Task: The code of OB100 forces the communication blocks to be restarted after a CPU restart. It sets M1.0 so that OB1 initializes the connection.

Fig.	4-5:	OB1	00
------	------	-----	----

OB100 : Title:
Comment:
Network 1: Enable function blocks
Comment:
M0.1 M0.1 (s)
Network 2 : Force restart of communication block
Comment:
M1.0 M1.0
(s)

4.3.3 FB100

What is FB100? This is a function block.

FB100 : Title:
Comment:
Network 1: Clear SEND BUSY-Bit to start initial communication
Comment:
#INIT_COM #SND_BUSY
Network 2: Start AG_SEND function with rising edge (clock marker MB10)
Start AG_SEND function with the rising edge of the clock marker if the AG_SEND function is not BUSY. The ACT input parameter of the AG_SEND function is triggered with a pulse and BUSY is set as long as the AG_SEND function is not completed! M10.1 is a memento that is set to 1 and 0 cyclically each 0,1s.
M10.1 #EDGE_CTRL #SND_BUSY #ACT_SEND (P) // () #SND_BUSY (s) //

Fig. 4-6: FB100

Task: The first network tests if the connection is initialized and resets the SND_BUSY flag so that no communication action can be done.

The second network tests the M10.1.

M10.1 is a clock memory. M10.1 stands for 100 ms. Each 100 ms the status of M10 bit 1 is changing (0/1).

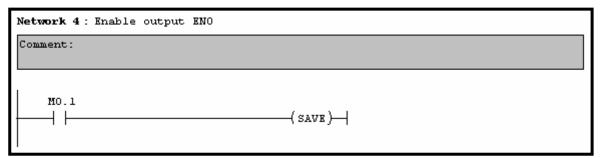
So this network is determining the cycle of the communication between the **S7** and the **MSX-Box**.

I M10.1 is equal to 1, communication can be done because SND_BUSY and ACT_SEND are set.

Task: Is seen in the previous network, ACT_SEND and SND_BUSY are set. Both are the input parameters of the FC10 (READ_DIG_INPUT). Here FC10 is called.

Network 3: Title	:		
- Send the teleg	gram to write all di	gital output	
	"READ_ DIG_INPUT"		
EN	- ENO		-
#ACT_SEND - ACT	_SEND SND_DONE	-#SND_DONE	
#SND_BUSY - SND	_BUSY SND_ERR	-#SND_ERR	
	SND_ STAT_SAVE	#SND_ -STAT_SAVE	
	RCV_ STAT_SAVE	#RCV_ -STAT_SAVE	
	RCV_LEN_ SAVE	#RCV_LEN_ -SAVE	

Task: Once FC10 returns, FB100 return value is saved.



4.3.4 FC10

What is FC10: In our sample, FC10 is the function to read all digital input status.

Task: At first the **S7** sends the MODBUS telegram to call MODBUS TCP function FC3 (Read multiple registers).

The telegram to send is predefined in a DB structure. Here DB101.

FC10 : Title:					
Comment:					
Network 1: In	Network 1: Invoke AG_SEND function block				
Ask for read	all digital inpu	uts of the APCI-1500.			
	"AG_SEND" EN	ENO			
#ACT_SEND -	ACT	DONE #SND_DONE			
1-	ID	ERROR #SND_ERR			
W#16#100-	LADDR S	STATUS #SND_STAT			
"SEND_ BUFFER". REQUEST_					
READ -	SEND				
12 -	LEN				

Fig. 4-7: FC10

ACT: ACT_SEND is an input parameter of the FC10 function. It was previously set. This enables sending data, it has to be seen as a trigger.

ID: 1 is the number of the communication identifier.

LADDR: This value W#16#100 is the value of the TCP Properties window:

Properties - TCP connection	
Options Overview General Information	Status Information Addresses Block Parameters
(D Pee): 0001 A050 Mane: Liaison TCP1 Via (DP: 009 343-1 Lean - (R0/54)	1-10 WE15E0100-LADOR
Boule C Active connection establishment Use ETP protocol	
OK	Cancel Help

SEND: This is a pointer on the data to send. Here "SEND_BUFFER".REQUEST_READ. In fact "SEND_BUFFER".REQUEST_READ is another name for an array defined in DB101.

This array contains the telegram to send.

DB101, REQUEST_READ_Array:

DB101

lddress	Name	Туре	Initial value	Comment	
0.0		STRUCT			
+0.0	REQUEST_READ	ARRAY[011]		ModbusTCP request:	Read all digital inputs on the fisrt board.
*1.0		BYTE			
+12.0	REQUEST_WRITE	ARRAY[014]		ModbusTCP request:	Write all digital outputs on the fisrt board.
*1.0		BYTE			
+28.0	BUFF	INT	0	Temp value.	
=30.0		END_STRUCT			
REQU	VEST_REAL) Array			
~~	VEST_REAL	Array	Initial value	Actual value	Comment
Address	1	Туре	Initial value	Actual value B#16#0	
Address	Name	Type Byte			
Address 0.0 1.0	Name REQUEST_READ[0]	Type Byte Byte	B#16#0	B#16#0	
Address 0.0 1.0 2.0	Name REQUEST_READ[0] REQUEST_READ[1]	Type BYTE BYTE BYTE	B#16#0 B#16#0	B#16#0	
Address 0.0 1.0 2.0 3.0	Name REQUEST_READ(0) REQUEST_READ(1) REQUEST_READ(2)	Type Byte Byte Byte Byte	B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0	
Address 0.0 1.0 2.0 3.0 4.0	Name REQUEST_READ[0] REQUEST_READ[1] REQUEST_READ[2] REQUEST_READ[3]	Type BYTE BYTE BYTE BYTE BYTE	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0 B#16#0	
Address 0.0 1.0 2.0 3.0 4.0 5.0	Name REQUEST_READ(0) REQUEST_READ(1) REQUEST_READ(2) REQUEST_READ(3) REQUEST_READ(4)	Type BYTE BYTE BYTE BYTE BYTE BYTE	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	
Address 0.0 1.0 2.0 3.0 4.0 5.0 6.0	Name REQUEST_READ(0) REQUEST_READ(1) REQUEST_READ(2) REQUEST_READ(3) REQUEST_READ(4) REQUEST_READ(5)	Type BYTE BYTE BYTE BYTE BYTE BYTE BYTE	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	
Address 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0	Name REQUEST_READ(0) REQUEST_READ(1) REQUEST_READ(2) REQUEST_READ(3) REQUEST_READ(4) REQUEST_READ(5) REQUEST_READ(6)	Type BYTE BYTE BYTE BYTE BYTE BYTE BYTE BYTE	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	
Address 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0	Name REQUEST_READ(0) REQUEST_READ(1) REQUEST_READ(2) REQUEST_READ(3) REQUEST_READ(4) REQUEST_READ(5) REQUEST_READ(6) REQUEST_READ(7)	Type BYTE BYTE BYTE BYTE BYTE BYTE BYTE BYTE	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#6 B#16#3	Comment ModbusTCP request: Read all digital input
Address 0.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0	Name REQUEST_READ(0) REQUEST_READ(1) REQUEST_READ(2) REQUEST_READ(3) REQUEST_READ(4) REQUEST_READ(5) REQUEST_READ(6) REQUEST_READ(7) REQUEST_READ(8)	Type BYTE BYTE BYTE BYTE BYTE BYTE BYTE BYTE	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#0	B#16#0 B#16#0 B#16#0 B#16#0 B#16#0 B#16#6 B#16#8 B#16#9 B#16#0	

Telegram description

B#16# stand for a hexadecimal byte value.

B#16#0

B#16#0

B#16#<mark>0</mark>

B#16#0

B#16#0

B#16#6

B#16#0

B#16#3

B#16#0

B#16#0

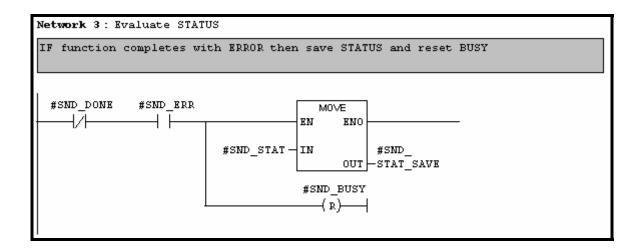
B#16#0	
B#16#1	
00 00 06	is the header. 06 indicate that the body is 6 bytes length.
0	is the slave ID.
3	function code.
00	reference number.
01	word count.
Mean: Rea	ad 1 word up reference 0 in the slave 0.
EN.	12 is the number of bute to send. The talegroup in

LEN: 12 is the number of byte to send. The telegram in *REQUEST_READ* means 12 byte length.

Task: If the communication succeeds, reset the SND_BUSY flag to enable a future communication. Here the telegram was sent without errors.

Network 2 : Function DONE	
If AG_SEND has completed reset BUSY	
#SND_DONE #SND_BUSY	#SND_BUSY (R)

Task: If an error occurred, save the error status and the SND_BUSY flag to enable a future communication.

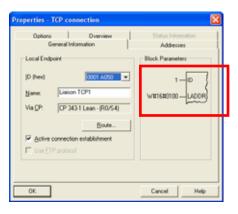


Task: Before an FC3 MODBUS request has been done. Now it is time to get reading values!

Network 4 : Ir	nvoke AG_RECV function	block			
Get the state	Get the status of all digital inputs.				
	"AG_RECV"]			
	EN ENC				
1 -	-ID NDI	R #RCV_NDR			
W#16#100-	LADDR ERROI	A -#RCV_ERR OUT: BOOL			
P#DB102.	STATUS	==#RCV_STAT			
DBX0.0					
BYTE 11-	RECV LEI	I-#RCV_LEN			

ID: 1 the communication ID like the AG_SEND function.

LADDR: This value W#16#100 is the value of the TCP Properties window:



RECV: This is a pointer on the place in which the response has to be saved, and the number of byte to read. Here the place used to save data is an array of 11 byte located in DB102.

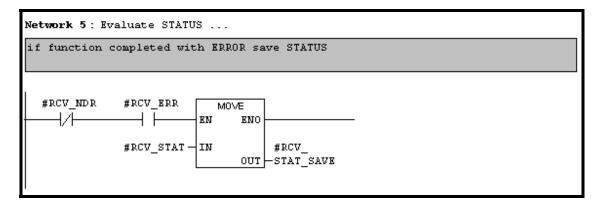
			DB102		
Address	Na	ume	Туре	Initial value	Comment
0.0			STRUCT		
+0.0	Π	RESPONSE	ARRAY[010]		
*1.0	11		BYTE		
=12.0			END STRUCT		

As you can find it in the OPEN MODBUS TCP description, the response telegram is:

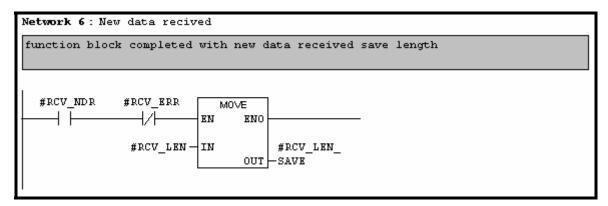
Telegram description
B #16# stand for a hexadecimal byte value.
B#16#0
B#16# <mark>0</mark>
B#16# <mark>0</mark>
B#16#0
B#16#0
B#16#6

B#16#0	
B#16# <mark>3</mark>	
B#16#2	
B#16#X	
B#16#X	
00 00 06	is the header. 06 indicate that the body is 6 bytes length.
0	is the slave ID.
3	function code.
2	number of byte constituting the data words.
XX	one word. XX is the value of the read reference.
Mean: Read 1 word (value XX) in reference 0 in the slave 0.	

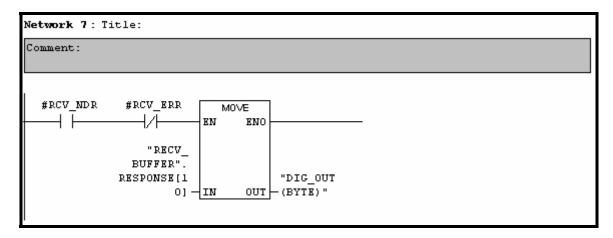
Task: If errors by receiving the telegram save the error status. Please note that in this sample, the telegram viability is NOT tested. But you are free to do so.



Task: If no errors by receiving the telegram save the number of reading data. Please note that in this sample, the telegram viability is NOT tested. But you are free to do so.



Task: In order to see the reading value, the first 8 byte of this value is written on the digital output of the **S7**. So when no error has occurred in the communication, the status of the 8 first digital inputs of the **APCI-1500** can be seen on the **S7** digital outputs LEDs.



"RECV_BUFFER".RESPONSE[10] is byte 11 of the reading telegram. As you can see it above, this byte is the LSB of the word corresponding of the read value.

5 COMMENTS

5.1 S7 programs

5.1.1 Program execution speed

- It is defined by the M10.1 "Clock memory byte" (memento). This clock memory is used to define an execution cycle. It is set on 100 ms. This is the minimal cyclic time that can be set with this memento. Please refer to Step7 that helps to know how to change the "Clock memory byte" cycle.
- A **S7** has no real cycle time that can be set. The cycle can be defined with a memento like in samples.