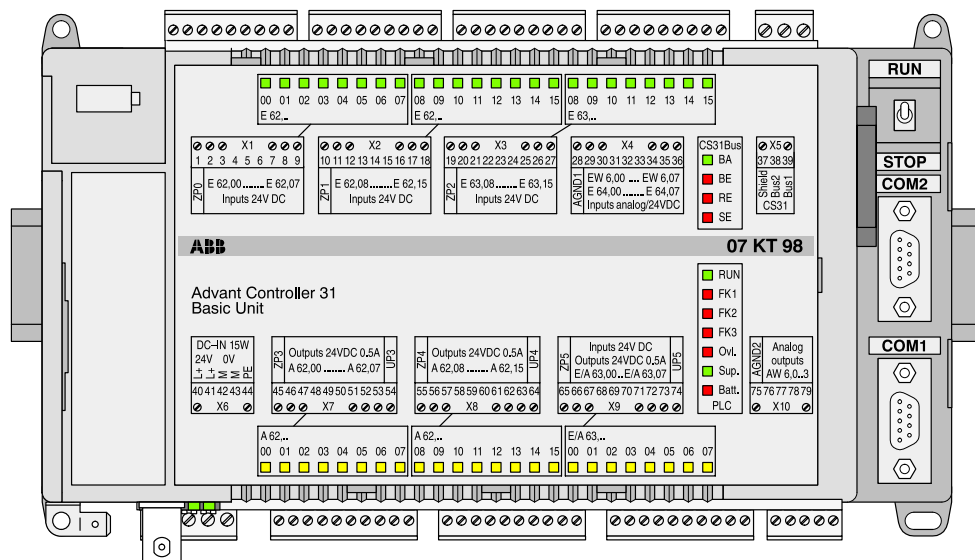


In order to easier find the different documentations, they are separated by coloured paper sheets.

System technology 90 series	Colour of paper sheet at the beginning
Basic units	red
Internal couplers	green
External couplers	blue



Contents

1	System technology of the basic units - Introduction	1-1
2	Operands of the basic units	2-1
2.1	Inputs / %I area	2-2
2.2	Outputs / %Q area	2-4
2.3	Operands in the %M area	2-6
2.4	Finding out absolute addresses of operands with the address operator ADR	2-7
2.5	System constants	2-10
2.5.1	Overview of System constants:	2-10
2.5.2	PLC application mode	2-12
2.5.3	Buffering / Initialization of flag areas	2-13
2.5.4	PLC reaction to class 3 errors	2-14
2.5.5	Setting the transmitting and receiving area of the slave PLC	2-15
2.5.6	Automatically initiated warm start after FK2 error	2-16
2.5.7	Configuration of the oscillators	2-16
2.5.8	Monitoring of the tasks	2-17
2.5.9	Monitoring of the floating-point functions	2-18
2.5.10	Setting the ARCNET timeout	2-18
2.5.11	Configuration of the digital inputs	2-19
2.5.12	Configuration of the operating mode for the high-speed counter	2-20
2.5.13	Configuration of the analog inputs	2-21
2.5.14	Configuration of the analog outputs	2-27
2.6	System and diagnosis flags	2-29
2.6.1	Oscillators	2-30
2.6.2	Interrupt from external networking interface	2-31
2.6.3	"Restart" detection	2-31
2.6.4	Error messages from PLC	2-31
2.6.5	Monitoring of the tasks	2-32
2.6.6	CS31 status word	2-32
3	Starting the PLC / program processing	3-1
3.1	Terms	3-1
3.2	Start of the user program	3-3
3.3	Initialization and backup of operands	3-5
3.3.1	Overview of initialization of variables	3-5
3.3.2	Notes for the declaration of buffered variables and constants	3-6
3.4	Processing of inputs and outputs in the multi-tasking system	3-8
4	Processing times	4-1
4.1	Terms	4-1

4.2	Program processing time	4-1
4.3	Set cycle time	4-2
4.4	Reaction time for digital signals	4-3
4.5	Cycle time monitoring	4-4
5	Addressing of the 07 KT 97 on the CS31 system bus	5-1
5.1	Introduction / Structure examples with 07 KT 97 as bus master	5-1
5.2	Recommended module addresses on the CS31 system bus	5-3
5.3	Address setting of the individual modules	5-4
5.4	07 KT 97 used as stand-alone basic unit	5-5
5.5	07 KT 97 used as bus master basic unit	5-5
5.6	Intelligent I/O-remote modules (basic units) as slave on the CS31 system bus	5-6
5.7	Special modules used as slave on the CS31 system bus	5-11
5.8	Complex structure examples including addresses	5-11
5.9	Module examples (slaves on the CS31 system bus)	5-11
6	I/O configuration	6-1
6.1	Purpose of the I/O configuration at I/O modules	6-1
6.2	Performing and reading the I/O configuration:	6-1
6.2.1	Performing and reading the I/O configuration via the user program	6-1
6.2.2	Reading of I/O configuration and diagnosis data at the remote module	6-2
7	Diagnosis	7-1
7.1	Introduction	7-1
7.2	Structure of the diagnosis	7-2
7.3	Troubleshooting by means of the LED displays on the basic unit	7-3
7.3.1	LEDs for CS31 system bus and bus interfacing	7-3
7.3.2	LEDs for user program and error display	7-4
7.3.3	LEDs for supply voltage and battery	7-4
7.3.4	LEDs for overload or short-circuit on at least one direct digital output	7-4
7.4	Troubleshooting on remote modules	7-5
7.4.1	Diagnosis displays on the remote modules	7-5
7.4.2	Diagnostic messages of the remote modules to the CS31 bus master	7-6
7.5	Acknowledgement of error messages in the remote modules	7-7
7.5.1	Example for an error message	7-7
7.5.2	LED displays on the bus master basic unit 07 KT 97	7-8
7.5.3	Reaction of the bus master basic unit 07 KT 97	7-8
7.5.4	Status word EW 07,15 in the basic unit 07 KT 97	7-8
7.5.5	Acknowledgement of the error flags in the basic unit 07 KT 97	7-8
7.5.6	Acknowledgement of the error flags in the remote modules	7-8
7.6	Error flags in the basic unit, error classification	7-9
7.7	Acknowledgement of error messages in the basic	7-10

7.8	Additional diagnosis functions	7-10
7.8.1	CS31 status word EW 07,15 / %IW1007.15	7-10
7.8.2	Capacity utilization of the basic unit and display of cycle time	7-10
7.8.3	Monitoring the floating-point functions	7-11
7.9	Meaning of the contents of the error word flags	7-13
7.9.1	FK1-Fatal errors	7-13
7.9.2	FK2-Serious errors	7-14
7.9.3	FK3-Light errors	7-15
7.9.4	FK4-Warning	7-17
7.10	Reaction of the bus master basic unit and the remote modules in case of errors	7-18
8	Serial interfaces COM1 and COM2	8-1
8.1	Operating modes of the serial interfaces	8-1
8.2	Basic features of the serial interfaces	8-1
8.3	Behaviour of the serial interfaces	8-2
8.3.1	Starting-up / Booting the PLC	8-2
8.3.2	Configuration by using function blocks in the user program	8-2
8.3.3	PLC RUN->STOP	8-3
8.3.4	Removing an existing connection between the pins 6 and 8 of the serial interface during a running user program	8-3
8.3.5	Automatic 907 AC 1131 login detection	8-4
8.4	Interface parameters	8-6
9	Programming and test	9-1
9.1	Programming system 907 AC 1131	9-1
9.2	Variants of programming the 07 KT 97 with 907 AC 1131	9-2
9.3	Programming via the serial interfaces	9-4
9.3.1	Serial driver "ABB RS232"	9-5
9.3.2	Serial driver "ABB RS232 Route"	9-6
9.4	Programming via ARCNET	9-9
9.4.1	ARCNET driver "ABB Arcnet"	9-10
9.4.2	ARCNET driver "ABB Arcnet Route"	9-11
9.4.3	ARCNET Driver "ABB Arcnet Route fast"	9-13
9.5	Programming 07 SL 97 slot PLC via the PCI interface	9-14
9.5.1	PCI Routing driver "ABB SL97"	9-15
9.5.2	PCI Routing Driver "ABB SL97 fast"	9-19
9.6	Programming via Ethernet (Tcp/Ip)	9-20
9.6.1	Ethernet driver "ABB Tcp/Ip Level 2"	9-21
10	List of function blocks	10-1
10.1	Overview of libraries	10-1
11	MODBUS RTU for the serial interfaces	11-1
11.1	Protocol description	11-1
11.2	MODBUS operating modes of the serial interfaces COM1 and COM2	11-2

11.3	Cable	11-3
11.4	MODBUS telegrams	11-3
11.5	Function block MODINIT	11-14
11.6	Function block MODMAST	11-14
11.7	Cross-reference list	11-14
11.8	MODBUS example program:	11-20
12	Index	I

1 System technology of the basic units - Introduction

The basic units 07 KT 95, 07 KT 96, 07 KT 97 and 07 KT 98 are programmed using the programming system 907 AC 1131. For this purpose, the new RT operating system is installed on the basic units.

The basic units 07 KR 91, 07 KT 92, 07 KT 93 and 07 KT 94 use a different operating system (EBS operating system). Therefore they **cannot** be programmed using the programming system 907 AC 1131.



Note:

This documentation is applicable for all categories of the basic units 07 KT 95, 07 KT 96, 07 KT 97 and 07 KT 98. When the term 07 KT 97 is given in the text, this text is also applicable for 07 KT 95, 07 KT 96 and 07 KT 98. Texts which are exclusively applicable for 07 KT 97 are expressly mentioned by a note.

2 Operands of the basic units

All the operands, which are supported by 907 AC 1131 are described in the documentation of the programming system 907 AC 1131. In this documentation the "address" operands (i.e. %I – for inputs, %Q – for outputs and %M – for addressable flags) which are mentioned in 907 AC 1131 are described in detail in the following.

All previous operands of the EBS operating system of the controllers 07 KR 91, 07 KT 92, 07 KT 93 and 07 KT 94 were imported into the 907 AC 1131. How to call the operands in 907 AC 1131 is described in the following. In the hardware documentation the previous designators are still used.

In the 907 AC 1131 all operands can be called bitwise (X), byte wise (B) and word wise (W). The double word operands can be called bitwise (X), byte wise (B), word wise (W) and double word wise (D).

Declaration of operands:

The declaration of the operands is done as following:

Symbol AT address : Type [:= initialization value]; (* comment *)
[.] - optional



Caution:

For multitasking the binary inputs and outputs for every task are byte wise cycle consistent, i.e. for instance the inputs E62,00-62,07 for task 1 and E62,08-E62,15 for task 2. This is not significant for programs with only one task.



Note:

With the software 907 AC 1131, Importfiles *.exp are automatically installed in the Library subdirectory. In this import files, important operands are already declared (see chapters Inputs, Outputs, System constants, system flags).

2.1 Inputs / %I area

Digital inputs:

E 00,00-E 61,15
%IW000-%IW061

E 62,00-E 63,15
%IW062-%IW063



Note:

The digital inputs E 63,00-E 63,07 / %IX63.0-%IX63.7 are the identifier for the 8 inputs of the digital I/Os.

E 64,00-E 64,07

%IW064

E 65,00-E 99,15
%IW065-%IW099

E 100,00-E 163,15
%IW100-%IW163

E 200,00-E 263,15
%IW200-%IW263

Digital inputs, CS31 remote modules:

%IB0-%IB123 %IX000.0-%IX061.15

Digital inputs of the basic unit 07 KT 97

%IB124-%IB127 %IX062.0-%IX063.15

Digital inputs of the basic unit 07 KT 97

(developed from the analog inputs EW 06,00...EW 6,07)

%IB128 %IX064.0-%IX064.7

reserved

%IB130-%IB199 %IX065.0-%IX065.15

reserved

%IB200-%IB327 %IX163.0-%IX163.15

reserved

%IB400-%IB527 %IX263.0-%IX263.15

Analog inputs:

EW 00,00-EW 05,15
%IW1000.0-%IW1005.15

EW 06,00-EW 06,07
%IW1006.0-%IW1006.07

EW 07,00-EW 07,14
%IW1007.0-%IW1007.14

EW 07,15
%IW1007.15

EW 08,00-EW 15,15
%IW1008.0-%IW1015.15

EW 16,00-EW 34,15
%IW1016.0-%IW1034.15

EW 100,00-EW 107,15
%IW1100.0-%IW1107.15

EW 200,00-EW 207,15
%IW1200.0-%IW1207.15

Analog inputs, CS31 remote modules

%IB1000.0.0-%IB1005.15.1 %IX1000.0.0-%IX1005.15.15

Analog inputs of the basic unit 07 KT 97

%IB1006.0.0-%IB1006.07.1 %IX1006.0.0-%IX1006.07.15

reserved

%IB1007.0.0-%IB1007.14.1 %IX1007.0.0-%IX1007.14.15

Status word for CS31 system bus

%IB1007.15.0-%IB1007.15.1 %IX1007.15.0-%IX1007.15.15

Analog inputs, CS31 remote modules

%IB1008.0.0-%IB1015.15.1 %IX1008.0.0-%IX1015.15.15

reserved

%IB1016.0.0-%IB1034.15.1 %IX1016.0.0-%IX1034.15.15

reserved

%IB1100.0.0-%IB1107.15.1 %IX1100.0.0-%IX1007.15.15

reserved

%IB1200.0.0-%IB1207.15.1 %IX1200.0.0-%IX1207.15.15

Inputs Line 1 and Line 2 (internal couplers):

Line 1:

%IW1.0-%IW1.1791 %IB1.0-%IB1.3583 %IX1.0.0-%IX1.1791.15

Line 2:

%IW2.0-%IW2.1791 %IB2.0-%IB2.3583 %IX2.0.0-%IX2.1791.15



Note:

Line 1 and Line 2 did not exist before.

Declaration examples for inputs:

The declaration of the operands is done as follows:

Symbol AT address : Type [:= initialization value]; (* comment *)

[.] - optional

Digital input E 62,00:

E62_00_Input0 AT %IX62.0 : BOOL; (* This is input 0 *)

Reading the digital inputs E62,00-E62,07 as a byte:

EB62_0_Byte0 AT %IB124 : BYTE; (* Input byte 62/0 *)

Reading the digital inputs E62,08-E62,15 as a byte:

EB62_1_Byte1 AT %IB125 : BYTE; (* Input byte 62/1 *)

Reading digital inputs E62,00-E62,15 as a word:

EW62_Word0 AT %IW62 : WORD; (* Input word 62 *)

Analog input EW06,00:

EW06_00_Ana0 AT %IW1006.0 : INT; (* This is analog input 0 *)

Calling bit 15 of the analog input EW06,00:

EW06_00_Bit15 AT %IX1006.0.15 : BOOL; (* Bit 15 of EW06,00 *)

Import file for inputs and outputs:

For the declaration of the inputs and outputs of the basic units, there are pre-produced import files available:

KT95_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 95
KT96_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 96
KT97_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 97
KT97_dea.exp	Digital onboard inputs and outputs of the basic unit 07 KT 97
KT98_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 98

The files are installed into the Library subdirectory of the software 907 AC 1131. In a project, they are inserted by means of the Resources menu into the object "Global Variables" with "Project / Options / Import".

Configuration of the digital inputs of the basic unit 07 KT 97:

The configuration of the digital inputs of the basic unit 07 KT 97 is carried out with the system constant KW85,00 / %MW3085.0, see chapter 'System constants'.

Configuration and measuring ranges of the analog inputs of the basic unit 07 KT 97:

The configuration of the analog inputs of the basic unit 07 KT 97 is carried out with the system constants KW86,00 / %MW1086.0..KW86,07 / %MW3086.7. In the chapter 'System constants' you can find the description of the configuration and selection of the measuring ranges of the analog input channels.

2.2 Outputs / %Q area

Digital outputs:

A 00,00-A 61,15 %QW000-%QW061	Digital outputs, CS31 remote modules %QB0-%QB123 %QX000.0-%QX061.15
A 62,00-A 63,07 : %QW062-%QW063	Digital outputs of the basic unit 07 KT 97 %QB124-%QB126 %QX062.0-%QX063.07
A 62,00 %QW062	High-speed counter, after activation direct output of „counter zero crossing“ %QB124 %QX062.0
A 64,00-A 99,15 %QW064-%QW099	reserved %QB128-%QB199 %QX064.0-%QX099.15
A 100,00-A 163,15 %QW100-%QW163	reserved %QB200-%QB327 %QX100.0-%QX163.15
A 200,00-A 263,15 %QW200-%QW263	reserved %QB400-%QB527 %QX200.0-%QX263.15

Analog outputs:

AW 00,00-AW 05,15 %QW1000.0-%QW1005.15	Analog outputs, CS31 remote modules %QB1000.0.0-%QB1005.15.1 %QX1000.0.0-%QX1005.15.15
AW 06,00-AW 06,03 %QW1006.0-%QW1006.03	Analog outputs of the basic unit 07 KT 97 %QB1006.0.0-%QB1006.03.1 %QX1006.0.0-%QX1006.03.15
AW 07,00-AW 07,15 %QW1007.0-%QW1007.15	reserved %QB1007.0.0-%QB1007.15.1 %QX1007.0.0-%QX1007.15.15
AW 08,00-AW 15,15 %QW1008.0-%QW1015.15	Analog outputs, CS31 remote modules %QB1008.0.0-%QB1015.15.1 %QX1008.0.0-%QX1015.15.15
AW 16,00-AW 34,15 %QW1016.0-%QW1034.15	reserved %QB1016.0.0-%QB1034.15.1 %QX1016.0.0-%QX1034.15.15
AW 100,00-AW 107,15 %QW1100.0-%QW1107.15	reserved %QB1100.0.0-%QB1107.15.1 %QX1100.0.0-%QX1107.15.15
AW 200,00-AW 207,15 %QW1200.0-%QW1207.15	reserved %QB1200.0.0-%QB1207.15.1 %QX1200.0.0-%QX1207.15.15

Outputs Line 1 and Line 2 (PROFIBUS):

Line 1: %QW1.0-%QW1.1791	%QB1.0-%QB1.3583	%QX1.0.0-%QX1.1791.15
Line 2: %QW2.0-%QW2.1791	%QB2.0-%QB2.3583	%QX2.0.0-%QX2.1791.15



Note:

Line 1 and Line 2 did not exist before.

Examples for the declaration of outputs:

The declaration of the operands is done as follows:

Symbol AT address : Type [:= initialization value]; (* comment *)

[.] - optional

Digital output A 62,00:

A62_00_Output0 AT %QX62.0 : BOOL; (* This is output 0 *)

Analog output AW06,00:

AW06_00_Ana0 AT %QW1006.0 : INT; (* This is analog output 0 *)

Output of digital outputs A62,00-A62,07 as a byte:

AB62_0_Byte0 AT %QB124 : BYTE; (* Output byte 62/0 *)

Output of digitale outputs A62,08-A62,15 as a byte:

AB62_1_Byte1 AT %QB125 : BYTE; (* Output byte 62/1 *)

Output of digital outputs A62,00-A62,15 as a word:

AW62_Word0 AT %QW62 : WORD; (* Output word 62 *)

Import file for inputs and outputs:

For the declaration of the inputs and outputs of the basic units, there are pre-produced import files available:

KT95_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 95
KT96_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 96
KT97_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 97
KT97_dea.exp	Digital onboard inputs and outputs of the basic unit 07 KT 97
KT98_onb.exp	Onboard inputs and outputs of the basic unit 07 KT 98

The files are installed into the Library subdirectory of the software 907 AC 1131. In a project, they are inserted by means of the Resources menu into the object "Global Variables" with "Project / Options / Import".

Configuration and measuring ranges of the analog outputs of the basic unit 07 KT 97:

The configuration of the analog outputs of the basic unit 07 KT 97 is carried out with the system constants KW88,00 / %MW1088.0..KW88,03 / %MW3088.3. In the chapter 'System constants' you can find the description of the configuration and the measuring ranges of the analog output channels.

2.3 Operands in the %M area

Flags:

M 000,00-M 254,15 %MW000-%MW254	Binary flags %MB000.0-%MB254.1 %MX000.0-%MX254.15
M 255,00-M 255,15 %MW255	System flags %MB255.0-%MB255.1 %MX255.0-%MX255.15
M 256,00-M 279,15 %MW256-%MW279	System flags / reserved %MB256.0-%MB279.1 %MX256.0-%MX279.15
M 280,00-M 511,15 %MW280-%MW511	Binary flags %MB280.0-%MB511.1 %MX280.0-%MX511.15

Word flags:

MW 000,00-MW 253,15 %MW1000.0-%MW1253.15	Word flags %MB1000.0.0-%MB1253.15.1 %MX1000.0.0-%MX1253.15.15
MW 254,00-MW 255,15 %MW1254.0-%MW1255.15	Error messages %MB1254.0.0-%MB1255.15.1 %MX1254.0.0-%MX1255.15.15
MW 256,00-MW 259,15 %MW1256.0-%MW1259.15	System flags / reserved %MB1256.0.0-%MB1259.15.1 %MX1256.0.0-%MX1259.15.15
MW 260,00-MW 511,15 %MW1260.0-%MW1511.15	Word flags %MB1260.0.0-%MB1511.15.1 %MX1260.0.0-%MX1511.15.15

Double word flags:

MD 00,00-MD 63,15 %MD2000.0-%MD2063.15 %MB2000.0.0-%MB2063.15.3	Double word flags %MW2000.0.0-%MW2063.15.1 %MX2000.0.0-%MX2063.15.31
--	---

Word constants:

KW 00,00-KW 00,15 %MW3000.0-%MW3000.15	System constants %MB3000.0.0-%MB3000.15.1 %MX3000.0.0-%MX3000.15.15
KW 01,00-KW 79,15 %MW3001.0-%MW3079.15	Word constants %MB3001.0.0-%MB3079.15.1 %MX3001.0.0-%MX3079.15.15
KW 80,00-KW 89,15 %MW3080.0-%MW3089.15	System constants %MB3080.0.0-%MB3089.15.1 %MX3080.0.0-%MX3089.15.15

Double word constants:

KD 00,00 %MD4000.0 %MB4000.0.0-%MB4000.00.3	System constant %MW4000.0.0-%MW4000.00.1 %MX4000.0.0-%MX4000.00.31
KD 00,01-KD 23,15 %MD4000.1-%MD4023.15 %MB4000.1.0-%MB4023.15.3	Double word constants %MW4000.1.0-%MW4023.15.1 %MX4000.1.0-%MX4023.15.31

Steps:

S 000,00-S 255,15 %MW5000-%MW5255	Steps (for project adoption from 907 PC 331) %MB5000.0-%MB5255.1 %MX5000.0-%MX5255.15
---	--

2.4 Finding out absolute addresses of operands with the address operator ADR

The address operator ADR is described in the documentation of the programming system 907 AC 1131 (refer to volume 5). In this documentation only the peculiarities applicable for bit operands are described.

The address operator ADR provides the address of an operand in one double word DWORD (i.e. 32 bits). The address operator provides the address of the first byte of a variable (byte address). For the free definable variables, variables of the type BOOL are stored as byte. In the addressable flag area %M, for the inputs %I and the outputs %Q, variables of the type BOOL are stored as real bits. Therefore, the address operator ADR provides in this case the variable address in addition to the bit number.

Standard format (for BOOL not addressable, BYTE, INT, WORD, DINT, DWORD,...)

Bit	Assignment
00..27	Variable address (bits 0..27)
28	Variable identifier 0 – non-addressable variable or addressable non-boolean variable 1 – addressable boolean variable
29..31	Variable address (bits 29..31)

The bits 27..31 of the address are always FALSE.

Special format for addressable binary variables of the type BOOL (%MX, %IX, %QX)

Bit	Assignment
00..03	Bit number The given bit number may have the values 0...15dec (0...Fhex). If the variable address is used, the bit number may only range from 0...7 (size of one byte), i.e. for a bit number of ≥ 8 , the value 8 must be subtracted in order to obtain 0...7.
04..27	Variable address of the byte (bits 0..23)
28	Variable identifier 0 – non-addressable variable or addressable non-boolean variable 1 – addressable boolean variable
29..31	Variable address of the byte (bits 24..26)

The bits 27..31 of the address are always FALSE.

The addresses, supplied by the address operator, can serve as inputs for those function blocks, which require absolute addresses (such as COPY, WOS, WOL). If these blocks must be used for internal variables, it must be made sure that the variables have addresses succeeding one another. This is performed by declaration of ARRAYS and STRINGS. The blocks with address inputs consider the peculiarities for the bit operands.

Examples for absolute addresses:

In a program, the following variables may be declared:

```
PROGRAM ADR_ST
VAR
  byZaehler           : BYTE; (* Zykluszähler (interne Variable) *)
  dwADR0              : DWORD; (* Address of byZaehler *)
  dwADR1              : DWORD; (* Address of DUMMY_M *)
  dwADR2              : DWORD; (* Address of M0_0 *)
  dwADR3              : DWORD; (* Address of E62_0 *)
  dwADR4              : DWORD; (* Address of A63_0 *)
  dwADR5              : DWORD; (* Address of S0_0 *)
  dwADR6              : DWORD; (* Address of DUMMY_MW *)
  dwADR7              : DWORD; (* Address of MW0_0 *)
  dwADR8              : DWORD; (* Address of EW62_0 *)
  dwADR9              : DWORD; (* Address of AW63_0 *)
  dwADR10             : DWORD; (* Address of SW0 )
  dwADR11             : DWORD; (* Address of the 1st byte of the ARRAY
                                DUMMY_byARRAY *)
  byQB124             AT %QB124 : BYTE; (* A62,00..A62,07 as Byte *)
  M0_15               AT %MX0.15 : BOOL; (* M00,15 *)
  E62_5               AT %IX62.5 : BOOL; (* E62,05 *)
  A63_7               AT %QX63.7 : BOOL; (* A63,07 *)
  S0_2                AT %MX5000.2 : BOOL; (* S00,02 *)
  MW0_0               AT %MW0 : INT; (* M00,00..M00,15 as Word *)
  EW62_0              AT %IW62 : INT; (* E62,00..E62,15 as Word *)
  AW63_0              AT %QW63 : INT; (* A63,00..A63,15 as Word *)
  SW0                 AT %MW5000 : INT; (* S00,00..S00,15 as Word *)
  DUMMY_M             : BOOL; (* internal variable BOOL *)
  DUMMY_MW            : INT; (* internal variable INT *)
  DUMMY_byARRAY       : ARRAY[0..99] OF BYTE; (* ARRAY of 100 bytes internal
*)
END_VAR
```

(* In the program part, the addresses of the variables are allocated to the variables ADRx:)

```
byZaehler := byZaehler + 1;
byQB124 := byZaehler; (* byZaehler sent to outputs A62,00..A62,07 *)
dwADR0 := ADR(byZaehler);
dwADR1 := ADR(DUMMY_M);
dwADR2 := ADR(M0_15);
dwADR3 := ADR(E62_5);
dwADR4 := ADR(A63_7);
dwADR5 := ADR(S0_2);
dwADR6 := ADR(DUMMY_MW);
dwADR7 := ADR(MW0_0);
dwADR8 := ADR(EW62_0);
dwADR9 := ADR(AW63_0);
dwADR10 := ADR(SW0);
dwADR11 := ADR(DUMMY_byARRAY[0]);
```


The following addresses are supplied:

Variable	Address on basic unit		Remarks
	07 KT 95..97	07 KT 98	
ByZaehler	16#001C7090	16#00603090	Byte address
DUMMY_M	16#001C70C1	16#006030C1	Byte address
M0_15	16#11B8001F	16#3800001F	Bit address→Mask: 16#FFFFFF0 / Bit No: 15
E62_5	16#12F807C5	16#171007C5	Bit address→Mask: 16#FFFFFF0 / Bit No: 5
A63_7	16#12FC07E7	16#171407E7	Bit address→Mask: 16#FFFFFF0 / Bit No: 7
S0_2	16#11BE5402	16#38065402	Bit address→Mask: 16#FFFFFF0 / Bit No: 2
DUMMY_MW	16#001C70C2	16#006030C2	Byte address
MW0_0	16#001B8000	16#02800000	Byte address
EW62_0	16#002F807C	16#0071007C	Byte address
AW63_0	16#002FC07E	16#0071407E	Byte address
SW0	16#001BE540	16#02806540	Byte address
DUMMY_byARRAY	16#001C70C4	16#006030C4	Byte address

Evaluation of the addresses:

In the example, the flag M00_00 is addressed with **M0_15** as Bit and the flags M00_00...M00_15 are addressed with **MW0_0** as Words. If the address is represented bitwise, the connection described above can be realized:

PLC	Variable	Address HEX	Address BIN				
		Bit: 31	28 27	15	4	3	0
07 KT 97	M0_15:	16#11B8 001F	2#000 1 0001 1011 1000 0000 0000 0001 1111				
	Bit 28:=TRUE → Mask:	16#EFFF FFF0	2#111 0 1110 1111 1111 1111 1111 1111 0000				
	Address (*16):	16#01B8 0010	2#000 0 0001 1011 1000 0000 0000 0001 0000				
	Byte address:	16#001B 8001	2#000 0 0000 0001 1011 1000 0000 0000 0001				
	Bit 0..3: 1111 → Bit number := 15						
07 KT 97	MW0_0:	16#001B 8000	2#000 0 0000 0001 1011 1000 0000 0000 0000				
	Bit 28 FALSE → Byte adresse corresponds to the supplied address						
07 KT 98	M0_15:	16#3800 001F	2#001 1 1000 0000 0000 0000 0000 0001 1111				
	Bit 28:=TRUE → Mask:	16#EFFF FFF0	2#111 0 1111 1111 1111 1111 1111 1111 0000				
	Address (*16):	16#2800 0010	2#001 0 1000 0000 0000 0000 0000 0001 0000				
	Byte address:	16#0280 0001	2#000 0 0010 1000 0000 0000 0000 0000 0001				
	Bit 0..3: 1111 → Bit number := 15						
07 KT 98	MW0_0:	16#0280 0000	2#001 0 1000 0000 0000 0000 0000 0000 0000				
	Bit 28 FALSE → Byte adresse corresponds to the supplied address						

If the found addresses are compared to each other, it can be seen that the addresses differ between the basic units 07 KT 97 and 07 KT 98. However, using the address operator ADR, the user must not take care for the absolute addresses.

As it can be seen, the address operator supplies the same byte address for both the bit-wise and the word-wise access to the same memory. With the 07 KT 97, for MW0_0 the byte address 16#001B8000 is supplied and for M0_15 the byte address 16#001B8001 is supplied. This value is correct, because M0_15 is located on the byte of the flags M00_08..M00_15, i.e. one byte higher than M00_00..M00_07. With MW0_0, the supplied address of M00_00..M00_07 is correct.

2.5 System constants

The constants

KW 00,00 – KW 00,15 / %MW3000.0 - %MW3000.15 and

KW 80,00 - KW89,15 / %MW3080.0 - %MW3089.15

are reserved for the usage as system constants. It is also not allowed to use constants from this areas which are not yet seized for other purposes.

The system constants are declared in the export file Sy_const.exp. When creating a new project using "File / New", these files are automatically stored in the "Resources" menu and there under the "Global variables" object. Otherwise they can be read in with "Project / Import".

2.5.1 Overview of System constants:

KW 00,00 : %MW3000.0	Setting of PLC application mode, (Stand-alone PLC, Master PLC, Slave PLC)
KW 00,01 : %MW3000.1	Initialization: Binary flags area
KW 00,02 : %MW3000.2	Initialization: Word flags area
KW 00,03 : %MW3000.3	Initialization: Double word flags area
KW 00,04 : %MW3000.4	Initialization: Step chain flags area (project adoption)
KW 00,05 : %MW3000.5	No function (with EBS: Initialization of historical values)
KW 00,06 : %MW3000.6	No function (with EBS: Application mode of ser. interface COM1)
KW 00,07 : %MW3000.7	PLC reactions to class 3 errors
KW 00,08 : %MW3000.8	No function (with EBS: 07 KT 92/93 overload / short circuit)
KW 00,09 : %MW3000.9	No function (with EBS: Starting up the CS31 system)
KW 00,10 : %MW3000.10	Size of the transmitting area of the slave PLC
KW 00,11 : %MW3000.11	Size of the receiving area of the slave PLC
KW 00,12 : %MW3000.12	Automatically initiated warm start after FK2 errors
KW 00,15 : %MW3000.15	Deactivation of oscillators on M 255,00-M 255,06
KW 80,00 : %MW3080.0	NEW: Monitoring of the task with cycle time > 0
KW 80,01 : %MW3080.1	NEW: Monitoring of the task with cycle time = 0
KW 81,00 : %MW3081.0	NEW: Reserved for memory requirement from the HEAP
KW 81,08 : %MW3081.8	NEW as of run-time system V4.16: Reaction to floating-point exception
KW 82,00 : %MW3082.0	NEW as of run-time system V4.16: Setting ARCNET timeout
KW 82,01 : %MW3082.1	NEW as of run-time system V4.16: Reserved for setting the ARCNET baud rate
KW 83,00 : %MW3083.0	NEW as of run-time system V4.16: Reserved for simulation mode

KW 85,00 : %MW3085.0	Configuration of the digital inputs E62,00-E62,15
KW 85,01 : %MW3085.1	Configuration of the digital inputs E63,00-E63,15
KW 85,02 : %MW3085.2	Configuration of the operating modes of the high-speed counter
KW 86,00.: %MW3086.0	Configuration of the analoge input EW 06,00
KW 86,01 : %MW3086.1	Configuration of the analoge input EW 06,01
KW 86,02 : %MW3086.2	Configuration of the analoge input EW 06,02
KW 86,03 : %MW3086.3	Configuration of the analoge input EW 06,03
KW 86,04 : %MW3086.4	Configuration of the analoge input EW 06,04
KW 86,05 : %MW3086.5	Configuration of the analoge input EW 06,05
KW 86,06 : %MW3086.6	Configuration of the analoge input EW 06,06
KW 86,07 : %MW3086.7	Configuration of the analoge input EW 06,07
KW 88,00 : %MW3088.0	Configuration of the analoge output AW 06,00
KW 88,01 : %MW3088.1	Configuration of the analoge output AW 06,01
KW 88,02 : %MW3088.2	Configuration of the analoge output AW 06,02
KW 88,03 : %MW3088.3	Configuration of the analoge output AW 06,03
KD 00,00 : %MD4000.0	No function (with EBS: Setting the cycle time)

2.5.2 PLC application mode

Used as master PLC, slave PLC or stand-alone PLC

KW 00,00 / %MW3000.0:

Meaning of the initialization value of the constants:

- Master PLC on CS31 system bus: -1 (FFFF_H)
 - Stand-alone PLC: -2 (FFFE_H)
 - Slave PLC on CS31 system bus: CS31 module addresses 0-61 and 100-115
 - Range of values: -2, -1, 0-61, 100-115
 - Default value: -2 (Stand-alone PLC)
-

Examples:

Declaration as stand-alone PLC:

```
KW00_00_MAST_SLV AT %MW3000.0 : INT := -2; (* Stand-alone PLC *)
```

Declaration as master on the CS31 bus:

```
KW00_00_MAST_SLV AT %MW3000.0 : INT := -1; (* Master PLC *)
```

Declaration as Slave No. 5:

```
KW00_00_MAST_SLV AT %MW3000.0 : INT := 5; (* Slave PLC *)
```



Caution:

Changing the PLC application mode is carried out in three steps:

1. Change the system constant KW 00,00 in the PLC
2. Create boot project (flash user program)
3. Activate the new PLC application mode by:
 - initiating a warm start (see 1.2.1 "Terms") or
 - initiating a cold start (see 1.2.1 "Terms").

For the operating mode "Slave PLC on the CS31 system bus" the following applies (refer also to chapter 1.5.6 - Intelligent I/O remote modules (basic units) as slaves on the CS31 system bus):

- The value ranges for the address are 0..61 and 100..115. The highest permissible address depends on the size of both the set sending area and the set receiving area. The greater these two areas are chosen, the smaller is the highest permissible address.
- The slave basic unit can be used on the CS31 system bus in both the digital and the word area. When operated in word area, the sending and receiving data are on the channels 0...7 or 8...15. The selection is performed together with the address setting:
KW00_00 / %MW3000.0 := 0..5; 8..15 → Channels 0..7
KW00_00 / %MW3000.0 := 100..105; 108..115 → Channels 8..15
By adding 100 to the address, the upper channel range of 8...15 is configured.

2.5.3 Buffering / Initialization of flag areas

Initialization of binary flags

KW 00,01 / %MW3000.1:

Meaning of the initialization value **n** of the constants:

→Initialized binary flag areas (set to 0)

n = 0 (default) M 000,00-M 511,15 / %MX0.0-%MX511.15

n = 1-511 M n,00-M 511,15 / %MXn.0-%MX511.15

n < 0, n > 511 M 255,10-M 511,15 / %MX255.10-%MX511.15

Example:

KW 00,01 := 52 and battery is available:

KW00_01_INIT_M AT %MW3000.1 : INT := 52; (* Initialization of binary flags *)

Flags initialized with 0: M 52,00-M 511,15 / %MX52.0-%MX511.15

Flags buffered: M 00,00-M 51,15 / %MX0.0-%MX51.15

Initialization of word flags KW 00,02 / %MW3000.2:

Meaning of the initialization value **n** of the constants:

→Initialized word flag areas (set to 0)

n = 0 (default) MW 000,00-MW 511,15 / %MW1000.0-%MW1511.15

n = 1-511 MW n,00-MW 511,15 / %MW1000+n.0-%MW1511.15

n < 0, n > 511 no initialization

Example:

KW 00,02 := 52 and battery is available:

KW00_02_INIT_MW AT %MW3000.2 : INT := 52; (* Initialization of word flags *)

Flags initialized with 0: MW 52,00-MW 511,15 / %MW1052.0-%MW1511.15

Flags buffered: MW 00,00-MW 51,15 / %MW1000.0-%MW1051.15

Initialization of double word flags

KW 00,03 / %MD3000.3:

Meaning of the initialization value **n** of the constants:

→Initialized double word flag areas (set to 0)

n = 0 (default) MD 00,00-MD 63,15 / %MD2000.0-%MD2063.15

n = 1-63 MD n,00-MD 63,15 / %MD2000+n.0-%MD2063.15

n < 0, n > 63 no initialization

Example:

KW 00,03 := 52 and battery is available:

KW00_03_INIT_MD AT %MD3000.3 : INT := 52; (* Initialization of double word flags *)

Flags initialized with 0: MD 52,00-MD 63,15 / %MD2052.0-%MD2063.15

Flags buffered: MD 00,00-MD 51,15 / %MD2000.0-%MD2051.15

Initialization of step chains

KW 00,04 / %MW3000.4:

Meaning of the initialization value **n** of the constants:

→ Initialized step chains (set to step 0)

n = 0 (default) S 00,00-S 255,15 / %MW5000-%MW5255

n = 1-255 S n,00-S 255,15 / %MW5000+n-%MW5255

n < 0, n > 255 no initialization

Example:

KW 00,04 := 52 and battery is available:

KW00_04_INIT_S AT %MW3000.4 : INT := 52; (* Initialization of step chains *)

Flags initialized with 0: S 52,00-S 255,15 / %MW5052-%MW5255

Flags buffered: S 00,00-S 51,15 / %MW5000-%MW5051



Note:

The step flags S 00,00-S 255,15 / %MW5000-%MW5255 are required for adopting projects which were created using 907 PC 331. The step flags **can not** be used in the same way as within the EBS operating system.

2.5.4 PLC reaction to class 3 errors

PLC reaction to FK3 errors:

KW 00,07 / %MW3000.7:

Meaning of the initialization value **n** of the constants:

n = 0 (default) Errors are only reported

n < 0, >0 Errors are reported and the PLC program is aborted

Changing the system constant takes effect immediately.

Example:

KW 00,07 := 0

KW00_07_FK3_REAK AT %MW3000.7 : INT := 0; (* FK3 errors are only reported *)

2.5.5 Setting the transmitting and receiving area of the slave PLC

Size of the transmitting area of the slave PLC

KW 00,10 / %MW3000.10:

Meaning of the initialization value **n** of the constants:

The slave PLC may either be used on the CS31 system bus in the binary area or in the word area. The binary values are transmitted byte by byte. It is possible to set the number of bytes or words which are sent from the slave to the master PLC.

- For use in the binary area: Sending 0...15 bytes 0...15
- For use in the word area: Sending 0...8 words 100...108
- Range of values: 0...15 and 100...108
- Default value: 0

Changing the system constant takes effect:

- on the next warm start or cold start
-

Example:

```
KW 00,10 := 0
KW00_10_SLV_SEND AT %MW3000.10 : INT := 0; (* 4 bytes of data transfer *)
```

Size of the receiving area of the slave PLC

KW 00,11 / %MW3000.11:

Meaning of the initialization value **n** of the constants:

The slave PLC may either be used on the CS31 system bus in the binary area or in the word area. The binary values are transferred byte by byte. It is possible to set the number of bytes or words which are sent from the slave to the master PLC.

- For use in the binary area: Sending 0...15 bytes 0...15
- For use in the word area: Sending 0...8 words 100...108
- Range of values: 0...15 and 100...108
- Default value: 0

Changing the system constant takes effect:

- on the next warm start or cold start
-

Example:

```
KW 00,11 := 0
KW00_11_SLV_REC AT %MW3000.11 : INT := 0; (* 4 bytes of data transfer *)
```



Note:

The default setting is as follows:

Binary area, sending 4 bytes and receiving 4 bytes.

This is achieved using the default combination KW 00,10 = KW 00,11 = 0. The planned combination KW 00,10 = KW 00,11 = 4 has the same effect as the default combination.

The combination KW 00,10 = KW 00,11 = 100 (sending 0 words and receiving 0 words) is not allowed.

When used in the word area, the unused upper 8 channels of the address can be seized by an analog module, if necessary.

2.5.6 Automatically initiated warm start after FK2 error

Reaction to FK2 errors:

KW 00,12 / %MW3000.12:

Meaning of the initialization value **n** of the constants:

n = 0 (default) No automatically initiated warm start after FK2 errors

n = 1 Automatically initiated warm start after FK2 errors

Changing the system constant takes effect after the next warm start.

Example:

KW 00,12 := 0

KW00_12_SYSTEM AT %MW3000.12 : INT := 0; (* no automatically initiated warm start *)

2.5.7 Configuration of the oscillators

System flags of the oscillators:

KW 00,15 / %MW3000.15:

Meaning of the initialization value **n** of the constants:

- n = 0 (default) Oscillators are active at M 255,00 to M 255,06

- n = 1 Oscillators are redirected to and active at M 256,00 to M 256,06

- n = 2 or 3 No oscillators available

Example:

KW 00,15= 52 and battery is available:

KW00_15_BLINK AT %MW3000.15 : INT := 0; (* Oscillators are active *)

2.5.8 Monitoring of the tasks

Monitoring of the task with a cycle time > 0

KW 80,00 / %MW3080.0:

Meaning of the initialization value **n** of the constants:

- n = 0 (default) FK2 error after the watchdog exceeded the cycle time three times or exceeded the triple cycle time
 - n > 0 A pause period of 10 % of the cycle time is performed after the task is finished if the watchdog exceeded the cycle time three times or if an FK2 error occurred after the watchdog exceeded the value of n [in ms].
- Range of values: 0-32767 ms
– Default value: 0
-

Example:

KW 80,00= 1000 and the task is entered into the task configuration together with the cycle time:

```
KW80_00_TASK_ZYKL AT %MW3080.0 : INT := 1000; (* FK2 after time period *)
```

Cycle time violations are entered into the system flags MW259,1-MW259,3 / %MW1259.1-%MW1259.3.

Monitoring of the task with a cycle time = 0

KW 80,01 / %MW3080.1:

Meaning of the initialization value **n** of the constants:

- n = 0 (default) The watchdog initiates an FK2 error if a time period of 10 s is exceeded.
 - n > 0 The watchdog initiates an FK2 error if the cycle time [in ms], corresponding to the value of n, is exceeded.
- Range of values: 0-32767 ms
– Default value: 0
-

Example:

KW 80,01= 1000 and no task or a cycle time of 0 is entered into the task configuration:

```
KW80_01_TASK_ZYKL0 AT %MW3080.1 : INT := 1000; (* FK2 after time period *)
```

2.5.9 Monitoring of the floating-point functions

Reaction to floating-point exception

KW 81,08 / %MW3081.8:

Meaning of the initialization value **n** of the constants:

- | | |
|-------------------|--|
| - n = 0 (default) | If a floating-point exception occurs, an FK2 error is output. The PLC turns to STOP. Using the calling hierarchy, it is possible to find out where the error is located. |
| - n = 1 | If a floating-point exception occurs, no error is output. Using the function block FPUXINFO (in SystemInfo_S90_V42.LIB) it is possible to find out whether or not during the calculation a floating-point exception occurred. Thus, it can be decided either to continue work with default values or shut down machine intentionally. |
| - Default value: | 0 |
-

Example:

KW 81,08 = 1 und FB FPUXINFO in the user program:

KW81_08_FP_EXC AT %MW3081.8 : INT := 1; (* no FK2 floating-point error *)



Note: This function is available as of runtime system V4.16.

You can find further information in chapter:

7.8.3 Diagnosis / Additional diagnostic functions / Monitoring of the floating-point functions

2.5.10 Setting the ARCNET timeout

Setting ARCNET timeout

KW 82,00 / %MW3082.0:

Meaning of the initialization value **n** of the constants:

Bit	0	configures timeout ET2 of coupler 1
	1	configures timeout ET1 of coupler 1
	2	reserved for timeout ET2 of coupler 2
	3	reserved for timeout ET1 of coupler 2
	4	reserved for timeout ET2 of coupler 3
	5	reserved for timeout ET1 of coupler 3
	6	reserved for timeout ET2 of coupler 4
	7	reserved for timeout ET1 of coupler 4

Setting:

Bit 1	Bit 0	ET1	ET2	Meaning
0	0	1	1	max. network size is 2 km
0	1	1	0	
1	0	0	1	
1	1	0	0	for large networks

Default value: 00 (max. network size is 2 km)

Example:

KW 82,00 = 03 : very big networks

KW82_00 AT %MW3082.0 : INT := 3; (* large networks *)



Note:

This function is available as of runtime system V4.16.

2.5.11 Configuration of the digital inputs

Configuration of the digital inputs E 62,00-E 62,15:

KW85,00 / %MW3085.0:

Meaning of the initialization value n of the constants:		
Bit 00	configures input	E 62,00 / %IX62.0
01		E 62,01 / %IX62.1
...		...
14		E 62,14 / %IX62.14
15		E 62,15 / %IX62.15
Bit = 0	Input delay	= 7 ms (default value)
Bit = 1	Input delay	= 1 ms
Default value: 0 (all inputs with an input delay of 7 ms)		

Example:

KW 85,00= 255: E 62,00-E 62,07 – 1 ms and E 62,08-E 62,15 – 7 ms
KW85_00 AT %MW3085.0 : INT := 255; (* Input delay E62,00-E62,15 *)

Configuration of the digital inputs E63,00-E63,15:

KW85,01 / %MW3085.1:

Meaning of the initialization value n of the constants:		
Bit 00	configures input	E 63,00 / %IX63.0
01		E 63,01 / %IX63.1
...		...
14		E 63,14 / %IX63.14
15		E 63,15 / %IX63.15
Bit = 0	Input delay	= 7 ms (default value)
Bit = 1	Input delay	= 1 ms
Default value: 0 (all inputs with an input delay of 7 ms)		

Example:

KW 85,01= 255: E 63,00-E 63,07 – 1 ms and E 63,08-E 63,15 – 7 ms
KW85_01 AT %MW3085.1 : INT := 255; (* Input delay E 63,00-E 63,15 *)



Note:

The digital inputs E 63,00-E 63,07 / %IX63.0-%IX63.7 are the identifiers for the 8 inputs of the digital I/Os.

2.5.12 Configuration of the operating mode for the high-speed counter

Operating mode of the high-speed counter

KW85,02 / %MW3085.2:

Meaning of the initialization value **n** of the constants:

- 00 No counter active (defaultvalue)
- 01 Mode 1, one up-counter
- 02 Mode 2, one up-counter with enable input
- 03 Mode 3, two up/down counters
- 04 Mode 4, two up/down counters, the second counter counts the pulses on the falling edge
- 05 Mode 5, one up/down counter with an edge-triggered set input, active on the rising edge
- 06 Mode 6, one up/down counter with an edge-triggered set input, active on the falling edge
- 07 Mode 7, one up/down counter for 2-channel incremental encoders

The high byte is configured with 0.

- Default value: 0 (no counter active)

Example:

KW 85,02 = 1: Mode 1, one up-counter

KW85_02_COUNTER AT %MW3085.2 : INT := 1; (* Configuration high-speed counter *)

2.5.13 Configuration of the analog inputs

Operating mode of the analog inputs

KW86,0x / %MW3086.x:

KW 86,00 / %MW3086.0 configures analog input EW 6,00 / %IW1006.0.
KW 86,01 / %MW3086.1 configures analog input EW 6,01 / %IW1006.1.
KW 86,02 / %MW3086.2 configures analog input EW 6,02 / %IW1006.2.
KW 86,03 / %MW3086.3 configures analog input EW 6,03 / %IW1006.3.
KW 86,04 / %MW3086.4 configures analog input EW 6,04 / %IW1006.4.
KW 86,05 / %MW3086.5 configures analog input EW 6,05 / %IW1006.5.
KW 86,06 / %MW3086.6 configures analog input EW 6,06 / %IW1006.6.
KW 86,07 / %MW3086.7 configures analog input EW 6,07 / %IW1006.7.

Meaning of the initialization value **n** of the constants (hex value in the low byte):

- 00_H Analog input 0-10 V (defaultvalue)
- 01_H unused
- 02_H Digital input with 7 ms input delay
- 03_H Analog input 0...20 mA
- 04_H Analog input 4...20 mA
- 05_H Analog input -10...+10 V
- 06_H Analog input 0...5 V
- 07_H Analog input -5...+5 V
- 08_H Analog input Pt100 in 2-wire configuration -50...+400°C
- 09_H Analog input Pt100 in 3-wire configuration -50...+400°C*
- 0A_H Analog input 0...10 V differential inputs *
- 0B_H Analog input -10...+10 V differential inputs *
- 0C_H Analog input 0...5 V differential inputs *
- 0D_H Analog input -5...+5 V differential inputs *
- 0E_H Analog input Pt100 in 2-wire configuration -50...+70°C
- 0F_H Analog input Pt100 in 3-wire configuration -50...+70°C*

Hex value in the high byte, meaning of the bits **15 14 13 12 11 10 09 08:**

- 00_H Plausibility check and cut-wire and short-circuit monitoring (default value)
- 01_H Cut-wire and short-circuit monitoring
- 02_H Plausibility check
- 03_H No monitoring

Refer to "Measuring ranges of the analog input channels" for more detailed information on monitoring.

Changing the system constant takes effect:

– on the next warm start or cold start

*) For the operating modes "Pt100 with 3-wire configuration" and all configurations with differential inputs always two adjacent analog inputs belong together (e.g. EW06,00 and EW06,01).

In this case, both inputs are configured according to the desired operating mode. The lower address has to be the even address (EW06,00) and the next higher address (EW06,01) has to be the odd address.

The converted analog value is available on the higher address (EW06,01).

Examples

a) EW06,00 / %IW1006.0 as an analog input $\pm 10V$, standard monitoring

Configuration in KW86,00:

High byte: 00_H = Standard monitoring

Low byte: 05_H = $\pm 10V$

KW86,00: 0005H = +5 decimal

KW86_00_Ana0 AT %MW3086.0 : INT := 5; (* Configuration EW06,00 *)

b) EW06,02 / %IW1006.2 (even) and EW06,03 / %IW1006.3 (the next higher) as analog input Pt100, 3-wire, -50...+400°C, only cut-wire and short-circuit monitoring

Configuration in KW86,02 / %MW3086.2:

High byte: 01_H = Cut-wire and short-circuit monitoring

Low byte: 09_H = Pt100, 3-wire, -50...+400°C

KW86,02: 0109H = +265 decimal

KW86_02_Ana2 AT %MW3086.2 : INT := 265; (* Configuration EW06,02 *)

Configuration in KW86,03 / %MW3086.3:

High byte: 01_H = Cut-wire and short-circuit monitoring

Low byte: 09_H = Pt100, 3-wire, -50...+400°C

KW86,03: 0109H = +265 decimal

KW86_03_Ana3 AT %MW3086.3 : INT := 265; (* Configuration EW06,03 *)

The measured value is available in **EW06,03 / %IW1006.3**.

Measuring ranges of the analog input channels:

Resolution in the PLC system:

The measured values are converted with a resolution of 12 bits, i.e. 11 bits plus sign for voltage and 12 bits without sign for current. The ranges 0..5 V and $\pm 5 V$ are converted with 10 bits plus sign.

Examples:

<u>Measuring range</u>	<u>Range of converted numbers</u>
-10 V..0..10 V	-32760D..0..32760D 8008H..0000..7FF8H
0..20 mA	0..32760D 0000..7FF8H

Further details can be found in volume 2, chapter 5.1 „General information for the use of analog inputs and outputs“.

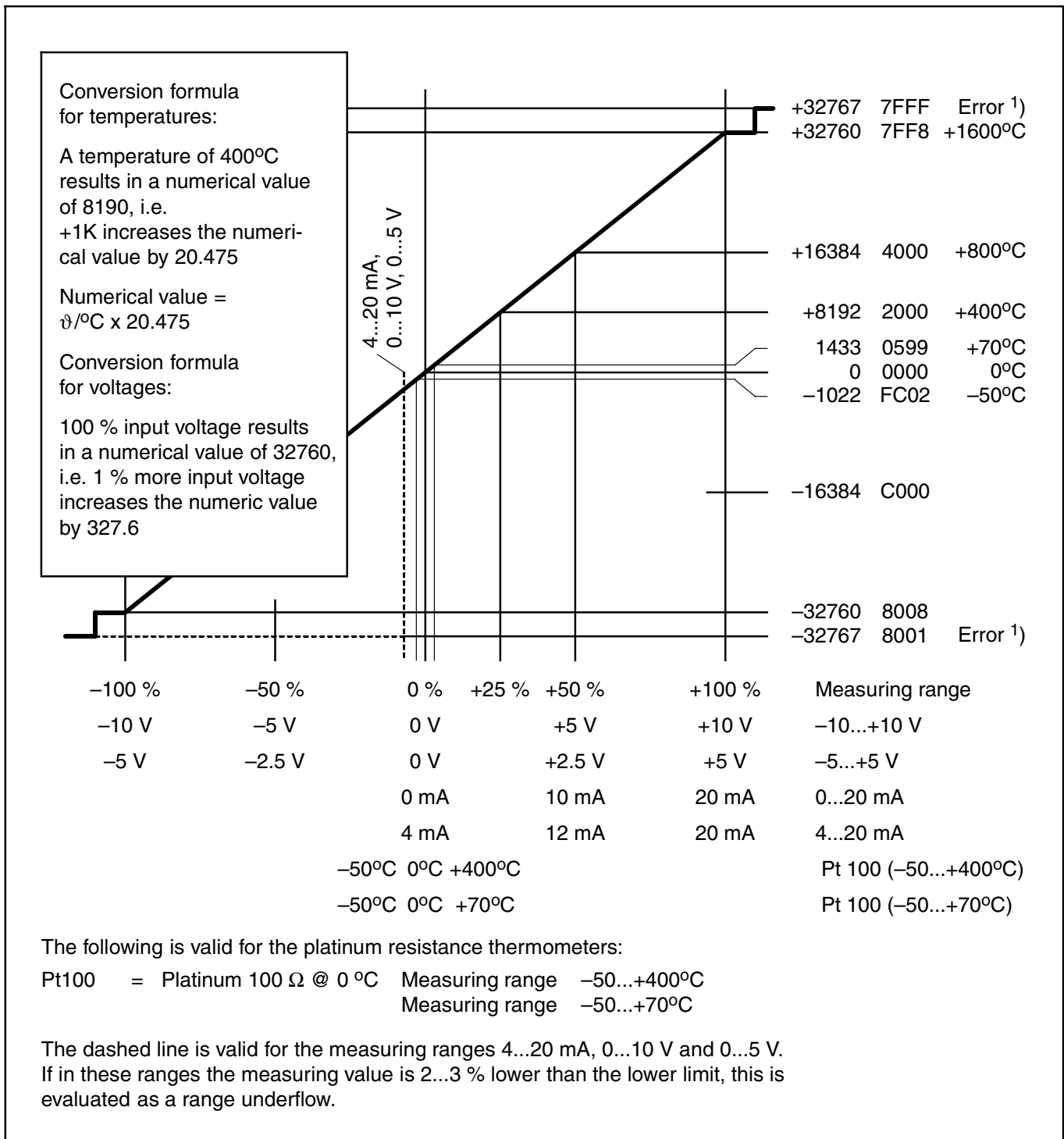
In order to make sure that **unused input channels** have a defined 0V level, they may be shorted to AGND. Unused inputs must be configured with **"unused"**.

The relationship between the analog input channels and the converted numbers is illustrated in the following figures.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	-100 %	50 %	25 %	12.5 %	6.25 %	3.13 %	1.56 %	0.78 %	0.39 %	0.20 %	0.10 %	0.05 %	0.02 %	0	0	0
Sign																
$\pm 10\text{ V}$	-10V	5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5 mV	2mV	0	0	0
$\pm 5\text{ V}$	-5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5mV	2mV	1mV	0	0	0
0...10 V		5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5mV	2mV	0	0	0
0...5 V		2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5mV	2mV	1mV	0	0	0
0...20 mA		10mA	5mA	2.5mA	1.25mA	625 μ A	313 μ A	156 μ A	78 μ A	39 μ A	20 μ A	10 μ A	5 μ A	0	0	0
4...20 mA		8mA	4mA	2mA	1mA	500 μ A	250 μ A	125 μ A	62.5 μ A	31.3 μ A	16 μ A	8 μ A	4 μ A	+ 4 mA offset		
Bit values	-32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Measuring ranges $\pm 10\text{ V}$, 0...10 V												11 bits resolution plus sign,				
Measuring ranges $\pm 5\text{ V}$, 0...5 V												10 bits resolution plus sign,				
Measuring ranges 0...20 mA, 4...20 mA												12 bits resolution without sign,				
the value range of -100...+100 % corresponds to the numbers 8008_H...7FF8_H (-32760...+32760),																
range overflow: 7FFF_H (32767), range underflow: 8001_H (-32767)																
open-circuit 4...20 mA: 8001_H (-32767)																
Remark: Independent of the resolution, all numbers are represented with 12 bits plus sign.																
Because of the results of internal calculations, all these bits can appear.																

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	-100 %	50 %	25 %	12.5 %	6.25 %	3.13 %	1.56 %	0.78 %	0.39 %	0.20 %	0.10 %	0.05 %	0.02 %	0.01 %	0.005 %	0
Sign																
Pt100	-1600°C	800°C	400°C	200°C	100°C	50°C	25°C	12.5°C	6.25°C	3.13°C	1.56°C	0.78°C	0.39°C	0.2°C	0.1°C	0
Bit values	-32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
The measuring range for Pt100 is 11 bits plus sign,																
the value range of -50...+400 °C corresponds to the numbers FC02_H...1FFE_H (-1022...+8190),																
the value range of -50...+70 °C corresponds to the numbers FC02_H...0599_H (-1022...+1433),																
range overflow / open-circuit: 7FFF_H (32767),																
range underflow / short-circuit of the sensor: 8001_H (-32767)																

Relationship between the measured values and the positions of the bits in the 16-bit word



Measuring ranges $\pm 10\text{ V}$ / $\pm 5\text{ V}$ / $0..10\text{ V}$ / $0..5\text{ V}$

Input voltages that exceed the measuring range cause the overflow numerical value of +32767. If the measured value is below the range, the underflow numerical value of -32767 is set. The input impedance is $> 100\text{ k}\Omega$.

Measuring range $4..20\text{ mA}$ (passive-type 2-pole sensors)

Input voltages that exceed the measuring range cause the overflow numerical value of +32767. If the measured value is below the range, the underflow numerical value of -32767 is set.

The input impedance is ca. $330\ \Omega$. The current input has a self-protecting mechanism. If the input current gets too high, the shunt is switched off and the value for range overflow is generated. About every second, the unit tries to switch on the shunt again. In this way the correct measurement will succeed after the current has reached a normal value again.

The trigger of the self-protecting mechanism is displayed by the red LED Ovl as long as the overload is present. In the PLC system an error message is then stored (FK4, error number 4).

The open-circuit monitoring begins below ca. 3 mA . The value of the range underflow is stored. If the open-circuit monitoring is configured, the open-circuit event is displayed by the red LED Ovl as long as it is present. In the PLC system an error message is stored (FK4, error number 9).

Measuring range $0..20\text{ mA}$ (active-type sensors with external supply voltage)

Input voltages that exceed the measuring range cause the overflow numerical value of +32767. If the measured value is below the range, the underflow numerical value of -32767 is set.

The input impedance is ca. $330\ \Omega$. The current input has a self-protecting mechanism. If the input current gets too high, the shunt is switched off and the value for range overflow is generated. About every second, the unit tries to switch on the shunt again. In this way the correct measurement will succeed after the current has reached a normal value again.

The trigger of the self-protecting mechanism is displayed by the red LED Ovl as long as the overload is present. In the PLC system an error message is then stored (FK4, error number 4).

Measuring ranges $\pm 10\text{ V}$ / $\pm 5\text{ V}$ / $0..10\text{ V}$ / $0..5\text{ V}$ as differential inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely earthed).

Since the earthing potential is not exactly the same as AGND1, it has to be measured bipolar in order to compensate measuring errors. Additionally, in case of single-pole configuration, AGND1 would be connected directly to the remote earth potential. This would cause inadmissible (and possibly dangerous) earthing loops.

In all configurations using **differential inputs** two adjacent analog inputs belong together (e.g.. EW06,00 / %IW1006.0 and EW06,01 / %IW1006.1).

The measured value is calculated by subtraction. The value of the channel with the lower address is subtracted from the value of the channel with the higher address.

The converted measured value is available on the odd address (e.g. EW06,01 / %IW1006.1).

Important:

The common mode input voltage range equals the measuring range of the single channel. I.e. that the signals, related to AGND, at the two involved inputs must not exceed this measuring range.

Input voltages that exceed the measuring range cause the overflow numerical value of +32767. If the measured value is below the range, the underflow numerical value of -32767 is set.

Measuring range $-50\text{ °C}..+400\text{ °C}$ and $-50\text{ °C}..+70\text{ °C}$ with Pt100 as temperature sensor in 2-wire configuration

When resistance thermometers are used, a constant current must flow through the measuring resistor in order to create the necessary voltage drop for the evaluation. For this purpose, the basic unit 07 KT 97 provides a constant current sink, which is multiplexed to the 8 analog channels.

Depending on the configured operating mode, the measured value is assigned linearly as follows:

<u>Range</u>	<u>assigned numerical value range</u>	
$-50\text{ °C}..400\text{ °C}$	-1022..+8190	(FC02 _H ..1FFE _H)
$-50\text{ °C}..70\text{ °C}$	-1022..+1433	(FC02 _H ..0599 _H)

The basic unit linearizes the Pt100 characteristic.

Temperatures, that exceed the measuring range cause the overflow numerical value of +32767. If the measured value is below the range, the underflow numerical value of -32767 is set.

In case of a detected open-circuit the numerical value of +32767 is set. If the sensor is short-circuited, the numerical value of -32767 is set.

If the open-circuit or short-circuit monitoring is configured, the detected error is displayed by the red LED Ovl as long as it is present. In the PLC system an error message is stored (FK4, error number 9).

In order to avoid error messages with unused analog inputs, it is useful, **not** to configure these channels for Pt100.

Measuring ranges $-50\text{ °C}..+400\text{ °C}$ and $-50\text{ °C}..+70\text{ °C}$ with Pt100 as temperature sensor in 3-wire configuration

In the operating mode "Pt100 in 3-wire configuration" two adjacent analog inputs belong together (e.g. EW06,00 / %IW1006.0 and EW06,01 / %IW1006.1).

For configuration, both inputs must be configured to the desired operating mode.

The constant current of the one channel flows through the Pt100 resistance sensor, the constant current of the other channel through one of the wires.

The basic unit calculates the measuring value from the two voltage drops and stores it under the odd address (e.g. EW06,01 / %IW1006.1).

Depending on the configured operating mode, the measured value is assigned linearly as follows:

<u>Range</u>	<u>assigned numerical value range</u>	
$-50\text{ °C}..400\text{ °C}$	-1022..+8190	(FC02 _H ..1FFE _H)
$-50\text{ °C}..70\text{ °C}$	-1022..+1433	(FC02 _H ..0599 _H)

The basic unit linearizes the Pt100 characteristic.

Temperatures, that exceed the measuring range cause the overflow numerical value of +32767. If the measured value is below the range, the underflow numerical value of -32767 is set.

In case of a detected open-circuit the numerical value of +32767 is set. If the sensor is short-circuited, the numerical value of -32767 is set.

If the open-circuit or short-circuit monitoring is configured, the detected error is displayed by the red LED Ovl as long as it is present. In the PLC system an error message is stored (FK4, error number 9).

In order to avoid error messages with unused analog inputs, it is useful, **not** to configure these channels for Pt100.

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. When doing so, they evaluate input voltages higher than ca. +7 V as signal TRUE.

The input signal delay is 7 ms. It cannot be changed by configuration. The inputs are not electrically isolated.

2.5.14 Configuration of the analog outputs

Operating mode of the analog outputs

KW88,0x / %MW3088.x:

KW 88,00 / %MW3088.0 configures analog output AW 6,00 / %QW1006.0.

KW 88,01 / %MW3088.1 configures analog output AW 6,01 / %QW1006.1.

KW 88,02 / %MW3088.2 configures analog output AW 6,02 / %QW1006.2.

KW 88,03 / %MW3088.3 configures analog output AW 6,03 / %QW1006.3.

Meaning of the initialization value **n** of the constants (hex value in the low byte):

- 00_H Analog output ±10 V (default value)

- 01_H unused

- 02_H Analog output 0...20mA

- 03_H Analog output 4...20 mA

Hexwert in the high byte: meaning of the bits **15 14 13 12 11 10 09 08**:

The high byte is reserved, has no meaning yet, can be configured with 00_H

Changing the system constant takes effect:

– on the next warm start or cold start

The measuring ranges of the analog outputs:

Resolution in the PLC system:

All analog output values are converted with a resolution of 12 bits, i.e. either 11 bits plus sign or 12 bits without sign.

Examples:

Range of numerical values	Output value
-32760D..0..32760D	-10 V..+10 V
8008H..0000..7FF8H	
0..32760D	0..20 mA
0000..7FF8H	

Further details can be found in volume 2, chapter 5.1 „General information for the use of analog inputs and outputs“.

Unused output channels may be left unconnected. They are configured as "**unused**".

The relationship between numerical values and the output analog signals is illustrated in the following figures.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	-100 %	50 %	25 %	12.5 %	6.25 %	3.13 %	1.56 %	0.78 %	0.39 %	0.20 %	0.10 %	0.05 %	0.02 %	0	0	0
	Sign															
± 10 V	-10V	5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5 mV	2mV	0	0	0
± 5 V	-5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5mV	2mV	1mV	0	0	0
0...10 V		5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5mV	2mV	0	0	0
0...5 V		2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5mV	2mV	1mV	0	0	0
0...20 mA		10mA	5mA	2.5mA	1.25mA	625 μ A	313 μ A	156 μ A	78 μ A	39 μ A	20 μ A	10 μ A	5 μ A	0	0	0
4...20 mA		8mA	4mA	2mA	1mA	500 μ A	250 μ A	125 μ A	62.5 μ A	31.3 μ A	16 μ A	8 μ A	4 μ A	+ 4 mA offset		
Bit values	-32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Measuring ranges ± 10 V, 0...10 V **11 bits resolution plus sign,**
Measuring ranges ± 5 V, 0...5 V **10 bits resolution plus sign,**
Measuring ranges 0...20 mA, 4...20 mA **12 bits resolution without sign,**
the value range of -100...+100 % corresponds to the numbers 8008_H...7FF8_H (-32760...+32760),
range overflow: 7FFF_H (32767), range underflow: 8001_H (-32767)
open-circuit 4...20 mA: 8001_H (-32767)

Remark: Independent of the resolution, all numbers are represented with 12 bits plus sign.
 Because of the results of internal calculations, all these bits can appear.

Relationship between numerical value and analog value

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	-100 %	50 %	25 %	12.5 %	6.25 %	3.13 %	1.56 %	0.78 %	0.39 %	0.20 %	0.10 %	0.05 %	0.02 %	0	0	0
	Sign															
± 10 V	-10V	5V	2.5V	1.25V	625mV	313mV	156mV	78mV	39mV	20mV	10mV	5 mV	2mV	0	0	0
0...20 mA		10mA	5mA	2.5mA	1.25mA	625 μ A	313 μ A	156 μ A	78 μ A	39 μ A	20 μ A	10 μ A	5 μ A	0	0	0
4...20 mA		8mA	4mA	2mA	1mA	500 μ A	250 μ A	125 μ A	62.5 μ A	31.3 μ A	16 μ A	8 μ A	4 μ A	+ 4 mA offset		
Bitwerte	-32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Measuring range ± 10 V **11 bits resolution plus sign,**
Measuring ranges 0...20 mA, 4...20 mA **12 bits resolution without sign,**
the value range of -100...+100 % corresponds to the numerical values 8008_H...7FF8_H (-32760...+32760),
Range overflow: 7FFF_H (32767), Range underflow: 8001_H (-32767)

Relationship between the output values and the position of the bits in the 16-bit word

2.6 System and diagnosis flags

The following areas are reserved for system flags and system flag words. It is also not allowed to use flags or flag words from this areas which are not yet seized for other purposes.

Flags: M255,00 – M279,15 %MX255.0 - %MX279.15
Flag words: MW254,00 – MW259,15 %MW1254.0 - %MW1259.15

The system and diagnosis flags are declared in the export file Fehler_M.exp and Auslastung.exp. When a new project is being created with "File / New", these files are automatically stored in the "Resources" menu under the "Global variables" object. Otherwise they can be read in with "Project / Import".

Overview of system flags:

%MX255.00 .. %MX255.06	Oscillators if %MW3000.15:=0
%MX255.09	Interrupt of external network interface
%MX255.10 .. %MX255.14	Error flag of the PLC
%MX255.15	Recognition of "New start" of the PLC
%MX256.00 .. %MX256.06	Oszillators if %MW3000.15:=1
%MW1254.00 .. %MW1255.15	Detailed information of the error flags of the PLC
%MW1259.00	Capacity utilization of the PLC
%MW1259.01 .. %MW1259.03	Display of exceeded cycle time of the PLC

2.6.1 Oscillators

Oscillators if %MW3000.15:=0:

M255,0-M255,6 /%MX255.0-%MX255.6:

M 255,00 : %MX255.0	Oscillator ca. 2 Hz
M 255,01 : %MX255.1	Oscillator ca. 1 Hz
M 255,02 : %MX255.2	Oscillator ca. 0.5 Hz
M 255,03 : %MX255.3	Oscillator with a period of ca. 1 minute (1/64 Hz)
M 255,04 : %MX255.4	Oscillator with a period of ca. 8 seconds (1/8 Hz)
M 255,05 : %MX255.5	Oscillator ca. 4 Hz
M 255,06 : %MX255.6	Oscillator ca. 8 Hz



Caution:

The oscillators are only active, if the system constant KW 00,15 / %MW3000.15 = 0 (see system constant KW 00,15 / %MW3000.15).

Oscillators bei %MW3000.15:=1:

M256,0-M256,6 /%MX256.0-%MX256.6:

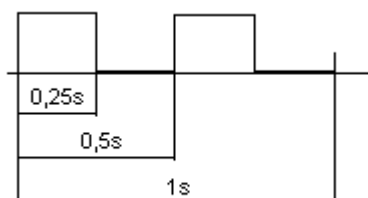
M 256,00 : %MX256.0	Oscillator ca. 2 Hz
M 256,01 : %MX256.1	Oscillator ca. 1 Hz
M 256,02 : %MX256.2	Oscillator ca. 0.5 Hz
M 256,03 : %MX256.3	Oscillator with a period of ca. 1 minute (1/64 Hz)
M 256,04 : %MX256.4	with a period of ca. 8 seconds (1/8 Hz)
M 256,05 : %MX256.5	Oscillator ca. 4 Hz
M 256,06 : %MX256.6	Oscillator ca. 8 Hz

See example: Oscillator 2 Hz (= 2 periods/s)



Caution:

The oscillators are only active, if the system constant KW 00,15 / %MW3000.15 = 1 (see system constant KW 00,15 / %MW3000.15).



Example: Oscillator 2 Hz (= 2 periods/s)

2.6.2 Interrupt from external networking interface

Interrupt external networking interface:

M255,9 / %MX255.9:

M 255,9 : %MX255.9 Interrupt from the external networking interface

A flag value of TRUE indicates that an interrupt was initiated by the external networking interface. The value must be set back to FALSE by the user.

2.6.3 "Restart" detection

"Restart" detection:

M255,15 / %MX255.15:

M 255,15 : %MX255.15 "Restart" detection (detection of the first cycle)

This binary flag can be used to detect the first program cycle after a program start. It is always initialized to "0" on each start of the user program, independent of the initialization settings predetermined by the system constant. Interrogating this flag in the user program and subsequently setting it to "1", allows to determine whether the user program was restarted or not.

2.6.4 Error messages from PLC

Error messages from the PLC:

M255,10-M255,14 / %MX255.10-%MX255.14:

M 255,10 : %MX255.10	Sum error message, indicates, that the PLC has detected an error
M 255,11 : %MX255.11	Error message FK1, fatal error, detailed information in MW 254,00-MW 254,07 / %MW1254.0-%MW1254.7
M 255,12 : %MX255.12	Error message FK2, serious error, detailed information in MW 254,08-MW 254,15 / %MW1254.8-%MW1254.15
M 255,13 : %MX255.13	Error message FK3, light error, detailed information in MW 255,00-MW 255,07 / %MW1255.0-%MW1255.7
M 255,14 : %MX255.14	Error message FK4, warning, detailed information in MW 255,08-MW 255,15 / %MW1255.8-%MW1255.15

Refer to chapter "Diagnosis" for a detailed description of the error messages.

Detailed information of the error flags of the PLC:

MW254,00-MW255,15 / %MW1254.00-%MW1255.15:

The detailed information of the error flags is contained in the area MW254,00-MW255,15 / %MW1254.0-%MW1255.15. Refer to chapter 1.6 "Diagnosis" for a detailed description of the error messages.

2.6.5 Monitoring of the tasks

Capacity utilization of the PLC:

MW259,00 /%MW1259.0:

The word flag MW259,0 / %MW1259.0 contains the current capacity utilization of the PLC in %. Refer to chapter 1.3 "Processing times" for more detailed information.

Indication of exceeded PLC cycle time:

MW259,01-MW259,03 /%MW1259.1-%MW1259.3:

Cycle time exceedings of the tasks are indicated in the word flags MW259,1-MW259,3 / %MW1259.1-%MW1259.3. Refer to chapter 1.3 "Processing times" for more detailed information.

MW 259,1 : %MW1259.1 Number of the task which exceeded the cycle time

MW 259,2 : %MW1259.2 Time in ms that calling of the task was performed too late

MW 259,3 : %MW1259.3 Number of missed task calls

(cumulative counter for all PLC user tasks)

2.6.6 CS31 status word

CS31 system bus status word:

EW 07,15 /%IW1007.15:

Bit 0 = 1 :No CS31 error of class 2 present

Bit 1 = 1 :PLC is added to the CS31 cycle (only relevant for use as slave)

Bit 2 = 1 :Time and date are valid

Bit 3 = 1 :Battery available

Bit 4...7 : Not used

Bit 8..15 :Maximum number of modules which were detected on the CS31 system bus up to the current point in time (only relevant for use as slave)

3 Starting the PLC / program processing

3.1 Terms

Cold start:

- All RAM memory modules are tested and erased.
- If there is no user program present in the Flash EPROM, all system constants are set to the default values (corresponds to the settings on delivery).
- If there is a user program present in the Flash EPROM, it is loaded into the RAM together with the system constants.
- The operating modes predetermined by the system constants are set.
- The CS31 system bus is reinitialized.

Initiating a cold start:

- Voltage ON/OFF if no battery is available or
- Menu item "Online/Cold start" in the programming system.

Warm start:

- All RAM memory modules are tested and erased except of the buffered operand areas and the RETAIN variables.
- If there is a user program present in the Flash EPROM, it is loaded into the RAM together with the system constants.
- The operating modes predetermined by the system constants are set.
- The CS31 system bus is reinitialized.

Initiating a warm start:

- Voltage ON/OFF if the battery is available or
- Menu item "Online/Reset" in the programming system.

STOP→RUN:

STOP→RUN means switching the RUN/STOP switch of the PLC from the STOP position to the RUN position.

If there is a user program in the RAM, it is started.

RUN→STOP:

RUN→STOP means switching the RUN/STOP switch of the PLC from the RUN position to the STOP position.

If there is a user program in the RAM, it will be aborted. All outputs will be set to 0 (FALSE).

STOP→START:

STOP→START means starting the user program with the programming system using the menu "Online/Start".

The user program contained in RAM is started.

START→STOP:

START→STOP means aborting the processing of the user program, contained in RAM, with the programming system using the menu "Online/Stop".

All outputs will be set to 0 (FALSE).

Download:

Download means loading the entire user program into the RAM of the PLC using the programming system's menu "Online/Load" or menu "Online/Log-in" (after the corresponding prompt during the log-in procedure).

Processing of the user program is aborted. In order to save the user program in Flash, the menu item "Online/Create boot project (flash)" must be performed after loading the program.

Online Change:

Online Change means loading of user program modifications into the RAM of the PLC using the programming system's menu "Online/Log-in" (after the corresponding prompt during the log-in procedure).

Processing of the user program is not interrupted. After sending the program modifications, the program is reorganized. This is shown by the text "REORG" in the status line of the programming system. During the reorganization, no further online change can be carried out. Saving the user program into Flash is not allowed as long as the reorganization is in progress. After that, saving can be performed using the menu "Online/Create boot project (flash)".

Buffering of data areas:

The buffering of data, i.e. the maintenance of data after voltage OFF/ON, is only possible when the battery is available. The following can be buffered completely or in parts:

- Binary flags
- Word flags
- Double word flags
- Step chains (for adopting projects generated with 907 PC 331)
- RETAIN variable

In order to buffer particular data, these data must be excluded from the initialization process.

3.2 Start of the user program

The start of the user program is carried out according to the following table. Precondition is, that there is a valid user program in Flash.

Action	No SMC card containing user program ¹⁾ inserted, RUN/STOP switch in RUN position	No SMC card containing user program ¹⁾ inserted, RUN/STOP switch in STOP position	SMC card containing user program ¹⁾ inserted, RUN/STOP switch in RUN position	SMC card containing user program ¹⁾ inserted, RUN/STOP switch in STOP position
Power ON or warm start or cold start	User program is loaded from Flash into the RAM and then started from the RAM.	No user program is loaded from the Flash. During LOGIN the message "No program in the PLC.." appears.	The user program is loaded from the SMC card to both the Flash and the RAM. Then it is started from the RAM.	The user program is loaded from the SMC card to the Flash. The RAM remains empty. During LOGIN the message "No program in the PLC.." appears.
STOP→RUN	The user program is started from the RAM.	The user program is started from the RAM. If there is no user program in RAM (see Power ON), it is loaded from the Flash and then started.	The user program is started from the RAM.	The user program is started from the RAM. If there is no user program in RAM (see Power ON), it is loaded from the Flash and then started.
STOP→START	The user program is started from the RAM..	The message "RUN/STOP switch in STOP position..." appears.	The user program is started from the RAM.	The message "RUN/STOP switch in STOP position..." appears.
Download ²⁾	The current user program in RAM of the PLC is aborted. The translated user program is loaded from the PC into the RAM of the PLC.	The translated user program is loaded from the PC into the RAM of the PLC.	The current user program in RAM of the PLC is aborted. The translated user program is loaded from the PC into the RAM of the PLC.	The translated user program is loaded from the PC into the RAM of the PLC.
Online Change ³⁾	The current user program in RAM is not interrupted. The changes of the user program are loaded from the PC into the PLC's RAM. The user program is reorganized and then processed.	The changes of the user program are loaded from the PC into the PLC's RAM. The user program is reorganized.	The current user program in RAM is not interrupted. The changes of the user program are loaded from the PC into the PLC's RAM. The user program is reorganized and then processed.	The changes of the user program are loaded from the PC into the PLC's RAM. The user program is reorganized.

Remarks:

- 1) The operating system version (runtime system) of the PLC, which was used to set up the SmartMedia card (SMC), must be equal to the version which is used to load the user program from the SMC. Otherwise, the user program is not loaded from the SMC.
- 2) After a download, the program is not automatically saved in the Flash. It is necessary, to carry out the menu item "Online/Create boot project (flash)". If the user program was not saved in Flash, the previous user program is loaded from the Flash after "Power ON". The program is started either with the RUN/STOP switch or with the programming system using "Online/Start".
- 3) After an Online Change, the program is not automatically saved in the Flash. In order to save the program, the menu item "Online/Create boot project (flash)" must be used after the reorganization has been terminated. During program reorganization as well as during the flashing procedure **a further Online Change must not** be carried out. If the user program was not saved in Flash, the previous user program is loaded from the Flash after "Power ON".

3.3 Initialization and backup of operands

3.3.1 Overview of initialization of variables

The initialization with 0 or to the initial value is performed with the program start.

For the operands in the % area, the areas (area parts) to be backed are set by system constants (see section "System constants"). If no battery is available or the system constants correspond to the delivery settings (default values) the data areas mentioned above are completely set to 0 after voltage OFF/ON.

If internal variables are to be backed, then they must be marked as „VAR_RETAIN“ with the declaration.



Note:

The internal variables only retain their order of sequence in case of an Online Change. If the program is translated, the order of sequence can change and (consequently) the backed values are wrong. See also chapter "Peculiarities Series 90".

The following table shows an overview of the initialization of the variables:

Action	VAR	VAR := value	VAR_RETAIN	VAR_RETAIN := value	%M	%M := value	%M buffered with KW0,x	%M buffered with KW0,x := value
Without battery								
Power ON	0	value	0	value	0	0	0	value
STOP→RUN	0	value	unch.	unch.	0	0	unch.	Wert
STOP→START	0	value	unch.	unch.	0	0	unch.	Wert
Download	0	value	unch.	unch.	0	0	unch.	Wert
Online Change	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unchanged
Reset (warm start)	0	value	0	value	0	0	0	value
Cold start	0	value	0	value	0	0	0	value
Battery available								
Power ON	0	value	unch.	unch.	0	0	unch.	value
STOP→RUN	0	value	unch.	unch.	0	0	unch.	value
STOP→START	0	value	unch.	unch.	0	0	unch.	value
Download	0	value	unch.	unch.	0	0	unch.	value
Online Change	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unchanged
Reset (Warmstart)	0	value	unch.	unch.	0	0	unch.	value
Cold start	0	value	0	value	0	0	0	value

unch. means unchanged



Note:

The area of the constants KW00,00-KW89,15 / %MW3000.0-%MW3089.15 and the double word constants KD00,00-KD23,25 / %MD4000.0-%MD4023.15 is stored in the Flash EPROM together with the user program and initialized with the initialization value when the controller is started.

3.3.2 Notes for the declaration of buffered variables and constants

In order to make sure, that the initialization/buffering of variables works correctly according to the table above, the following rules must be observed with the declaration.

Buffering internal variables:

The variables must be declared as **VAR_RETAIN** or **VAR_GLOBAL RETAIN**.

Example:

(* Declaration in the global variables lists *)

```
VAR_GLOBAL RETAIN
byVar : BYTE;
wVar : WORD;
rVar : REAL;
END_VAR
```

(* Declaration in the program *)

```
VAR RETAIN
byVar1 : BYTE;
END_VAR
```

Buffering of variables in the %M range:

The variables must be declared as VAR or VAR_GLOBAL with an address. By means of the system constants KW00_01..KW00_04 the ranges to be buffered are set.



Important:

The system constant KW00_02 **must not** be declared as VAR_GLOBAL CONSTANT and the variables **must not** be declared as VAR_GLOBAL RETAIN or VAR RETAIN.

Example:

Buffering the range MW00_00 / %MW1000.0 .. MW03_15 / %MW1003.15

(* Settin the range in the initialization value of the system constant MW00_02 / %MW3000.2 under Ressources / Global Variables / System constants

Important: DO NOT declare as VAR_GLOBAL CONSTANT *)

```
VAR_GLOBAL
KW00_02 AT %MW3000.2 : INT := 4;
END_VAR
```

(* Declaration of the variables as global variables *)

```
VAR_GLOBAL
MW00_00 AT %MW1000.0 : INT;
iV2 AT %MW1000.1 : INT;
..
MW03_15 AT %MW1003.15 : INT;
END_VAR
```

(* or declaration of the variables in the program *)

```
VAR
MW00_00 AT %MW1000.0 : INT;
iV2 AT %MW1000.1 : INT;
..
MW03_15 AT %MW1003.15 : INT;
END_VAR
```

Declaration of constants:

Constants are declared as **VAR_GLOBAL CONSTANT** or **VAR_CONSTANT**.

Example:

```
(* Declaration as global constants *)  
VAR_GLOBAL CONSTANT  
byConst_1 : BYTE := 1;  
END_VAR
```

```
(* Declaration in the program *)
```

```
VAR CONSTANT  
byConst_2 : BYTE := 2;  
END_VAR
```



Note:

As of version V5.0 of the 907 AC 1131 software, under "Project"=>"Options"=>"Translation options"=>"Replace constants" can be set, whether the constants should be treated like variables (i.e. being overwritable) or the value should be directly inserted in the code.

Declaration of constants in the %M range:

The constant ranges KW00,00-KW89,15 / %MW3000.0-%MW3089.15 and the double word constant ranges KD00,00-KD23,25 / %MD4000.0-%MD4023.15 are stored in the Flash together with the user program and initialized with the initialization value when the control system starts up.



Important:

The initialization of the constants does not work when they are declared as **VAR_GLOBAL CONSTANT** or **VAR CONSTANT**.

The constants

KW 00,00 – KW 00,15 / %MW3000.0 - %MW3000.15 and
KW 80,00 - KW89,15 / %MW3080.0 - %MW3089.15

are reserved as system constants. Constants within this ranges, which are not used yet, must under no circumstances be used for other purposes!

The declaration of the constants in the %M range is carried out with **VAR_GLOBAL** or **VAR** together with address and initialization value.

Example:

```
(* Declaration as global constant )  
VAR_GLOBAL  
iConst_10 AT %MW3001.0 : INT := 10;  
diConst_100 AT %MW4001.0 : DINT := 100;  
END_VAR
```

```
(* Declaration in the program *)
```

```
VAR  
iConst_20 AT %MW3010.0 : INT := 20;  
diConst_25 AT %MD4010.0 : DINT := 25;  
END_VAR
```

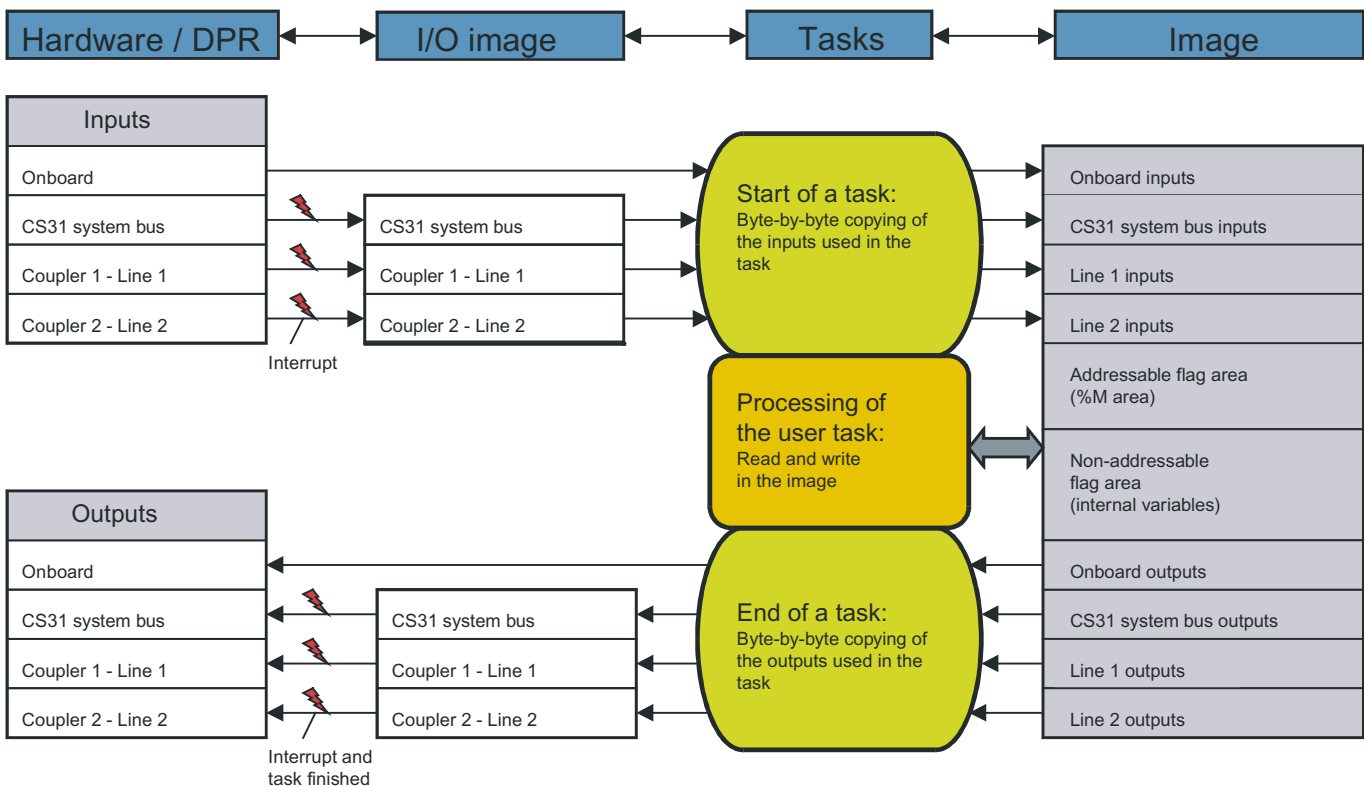
3.4 Processing of inputs and outputs in the multi-tasking system

In the following it is described, in which way the inputs and outputs are processed in the multi-tasking system.

The basic unit 07 KT 97 has the following types of inputs and outputs:

- **Onboard inputs and outputs:**
 - They are processed by the I/O processor and then provided for the working processor via the Dual-Port RAM (DPR).
- **Inputs and outputs on the CS31 system bus:**
 - They are processed by the CS31 processor and then provided for the working processor via the Dual-Port RAM (DPR).
- **Inputs and outputs of Line 1:**
 - They are processed by the coupler 1 (with 07 KT 97 R120 – PROFIBUS DP) and then provided for the working processor via the Dual-Port RAM (DPR).
- **Inputs and outputs of Line 2:**
 - They are processed by the coupler 2 (with 07 KT 97 R162 - PROFIBUS DP) and then provided for the working processor via the Dual-Port RAM (DPR).

The following figure shows the way the inputs and outputs are processed in the multi-tasking system.



The processing cycle has the following order of sequence:

Creating the input image:

Onboard inputs:

The Onboard inputs work without an additional input image.

Inputs on the CS31 system bus:

After the CS31 processor has processed all of the I/O modules, an interrupt is initiated at the working processor. In the interrupt service routine (ISR), the inputs are copied from the DPR to the input image of the working processor. If the outputs were updated by a task, they are copied from the output image to the DPR.

Inputs of Line 1 and Line 2:

If a coupler (1 or 2) has new data, an interrupt is initiated at the working processor. In the interrupt service routine (ISR), the inputs are copied from the DPR to the input image of the working processor. If the outputs were updated by a task, they are copied from the output image to the DPR.

Precondition is, that the coupler has a valid configuration.

Start of a task:

When starting a task, the **used inputs of the CS31 system bus as well as of the lines 1 and 2 are copied byte by byte** from the input image to the image. All of the onboard inputs are copied directly from the DPR of the I/O processor to the image. Byte by byte means, that, for instance, using `E00_00 / %IX0.0` the image of the inputs `E00_00 / %IX0.0...E00_07 / %IX0.7` are copied to the image.

Since only the inputs are copied, which are directly used in the task, the inputs cannot be read directly, if they have to be cycle-consistent.

Processing a task:

All the tasks work on the image, i.e. the inputs are read from the image and the outputs are written to the image. In online mode, the inputs and outputs of the image are displayed.

End of a task – Processing the output image:

At the end of a task, the **used outputs of the CS31 system bus as well as of the lines 1 and 2 are copied byte by byte** from the image to the output image. The onboard outputs are directly written from the image to the DPR of the I/O processor. Byte by byte means, that, for instance, using `A00_00 / %QX0.0` the image of the outputs `A00_00 / %QX0.0...A00_07 / %QX0.7` are copied from the image to the output image. The internal variables "Output image updated" are set for the CS31 processor and the couplers 1 and 2.

Writing the outputs:

Onboard outputs:

The onboard outputs work without an additional output image.

Outputs on the CS31 system bus:

With the next interrupt of the CS31 processor, the outputs are written from the output image to the DPR, and the variable "Output image updated" is reset.

Outputs of the lines 1 and 2:

With the next interrupt of the coupler 1 (or 2), the outputs are written from the output image of line 1 (or 2) to the DPR, and the variable "Output image updated" is reset.

I/O update task:

In order to update also the inputs and outputs which are not used in the tasks, all of the inputs and outputs are updated in the image using a task with a low priority (I/O update task). This task is only processed as long as no other user task is being processed.

4 Processing times

4.1 Terms

The most important times for the use of the basic unit 07 KT 97 with or without connected remote modules are as follows:

- The **reaction time** is the time between a signal transition at the input terminal and the signal response at the output terminal.
For digital signals, the reaction time consists of the input delay, the cycle time of the program processing and the bus transmission time, if the system is expanded by remote modules.
- The **cycle time** determines the time intervals after which the processor restarts the execution of the user program.
The cycle time has to be specified by the user. It should be greater than the program processing time of the user program plus the data transfer times and the related waiting times.
The cycle time is also the time base for some time-controlled functions, such as for the PID controller.
- The **program processing time** is the net time for processing the user program.

4.2 Program processing time

<u>Instructions</u>	<u>07 KT 95..97</u>	<u>07 KT 98</u>
---------------------	---------------------	-----------------

- **Binary instructions of the type:**

!M /M &M =M

!NM /NM &NM =NM

Processing time for 1000 instructions:	1.22 ms	0.32 ms
--	---------	---------

!M /M &M =SM

!NM /NM &NM =RM

Processing time for 1000 instructions:	1.46 ms	0.52 ms
--	---------	---------

- **Word instructions of the type:**

!MW +MW -MW =MW

!-MW -MW +MW =-MW

Processing time for 1000 instructions:	3.37 ms	0.90 ms
--	---------	---------

!MW *MW :MW =MW

!-MW *-MW :-MW =-MW

Processing time for 1000 instructions:	4.78 ms	1.14 ms
--	---------	---------

- **Mixed instructions**

- 65 % binary: !, /, &, =

- 20 % word: !, +, -, =

- 15 % word: !, *, :, =

Processing time for 1000 instructions:	2.18 ms	0.48 ms
--	---------	---------

4.3 Set cycle time

It is assumed that the processor always gets access in a moment with a *worst-case* condition.

The cycle time is stored in the task configuration and can be selected in steps of 1 ms. If the selected cycle time is too short, the processor will not be able to fulfil the tasks assigned to it every cycle. This will result in a time delay.

If this lack of time becomes too large over several cycles, the processor aborts the program execution and outputs an error (FK2).

Using some function blocks, such as the PID controller, the error-free execution depends on an exact timing sequence. Make sure that there is a larger time reserve.

The correct setting of the cycle time should be checked using the following procedure:

- Loading the user program into the basic unit.
- If the operating mode has been changed from standalone to bus master: Power ON or menu item "Online/Reset" in the programming software.
- Interrogation of the capacity utilization using the menu item "Online/Global Variable/Utilization".
- Changing the cycle time until the capacity utilization is below 80 %.

When setting the cycle time the following values are to be taken into account:

- Block copy time: Time for copying the input signals from the DPR to the I/O image:
 - digital inputs (E):
 - basic run time $28 \mu\text{s} + 3 \mu\text{s}$ per group of inputs (E)
 - + $21 \mu\text{s}$ for input (E) via the CS31 bus
 - analog inputs (EW):
 - basic run time $20\text{...}50 \mu\text{s} + 6 \mu\text{s}$ per group of inputs (EW)
 - 2 interface interrupts: $2 * 50 \mu\text{s}$
- Time for copying the input signals of the user task from the I/O image to the image memory
- Program processing time
- Time for copying the output signals of the user task from the I/O image to the image memory
- Block copy time: Time for copying the output signals from the I/O image into the DPR:
 - digital outputs (A):
 - basic run time $33 \mu\text{s} + 5 \mu\text{s}$ per group of outputs (A)
 - + $30 \mu\text{s}$ for output (A) via the CS31 bus
 - analog outputs (AW):
 - basic run time $20\text{...}32 \mu\text{s} + 6 \mu\text{s}$ per group of outputs (AW)
 - 2 interface interrupts: $2 * 50 \mu\text{s}$
- When using the setting Master or Slave, one interrupt from the CS31 bus must be added for each CS31 bus cycle ($36 \mu\text{s}\text{...}330 \mu\text{s}$):
 - basic run time $36 \mu\text{s} + 8 \mu\text{s}$ per group of inputs (EW)
 - + $8 \mu\text{s}$ per group of outputs (AW) + $30 \mu\text{s}$ for E + $40 \mu\text{s}$ for A
- Receiving/Sending interrupts from ARCNET telegrams within the cycle time

- Receiving/Sending interrupts from the serial interface within the cycle time
- Task changes
- Runtime of watchdog task

4.4 Reaction time for digital signals

Direct inputs and outputs:

- Delay of the digital inputs (E), configurable 7 ms / 1 ms
- Cycle time of the I/O processor, approx. 1 ms
- 2 * PLC cycle time (configurable by the user)
- Cycle time of the I/O processor, approx. 2 ms
- Delay of the digital outputs (A) (negligible)

Inputs and outputs via the CS31 bus:

- Delay of the digital remote modules, usually 8 ms
- Basic time of CS31 bus (2 ms) + 2 * Sum of the bus transmission times of the remote modules (refer to volume 2 for the bus transmission times of the remote modules).
- Interrupt CS31 bus (see cycle time: 36 μ s...330 μ s)
- 2 * PLC cycle time configurable by the user)
- Basic time of CS31 bus (2 ms) + Sum of the bus transmission times of the remote modules
- Interrupt CS31 bus (see cycle time: 36 μ s...330 μ s)
- Delay of the digital outputs (A), usually < 1 ms

4.5 Cycle time monitoring

In order to monitor the cycle time and in particular to prevent infinite loops, the two system constants KW80,0 / %MW3080.0 and KW80,1 / %MW3080.1 are available. The mode of action is described in the following.

There are two different cases for an user program which runs with a set cycle time:

- The program end is reached but the cycle time is not met. In this case the task is able to monitor itself.
- The program end is not reached ("infinite loop"). The task is then stopped by the watchdog task. The watchdog task works with an interval of 200 ms. That means that this monitoring has an "inaccuracy" of 200 ms.

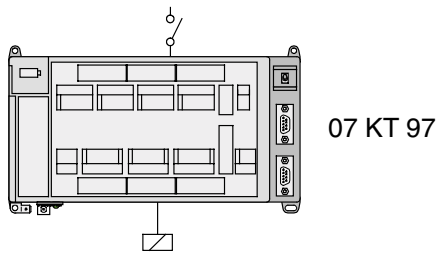
Also, a user program with a cycle time of 0 can be configured. This is the default setting with PLC_PRG and without task configuration. A cycle time of 0 means that the program runs "as fast as possible" and there is no definite time that has to be met. This program is only monitored by the watchdog.

If a cycle time error occurs, an FK2 error with the error number 131 is initiated. The detailed information of the FK2 error contains the task ID. All user programs go into the STOP state. In order to restart the program, a warm start or a cold start can be performed.

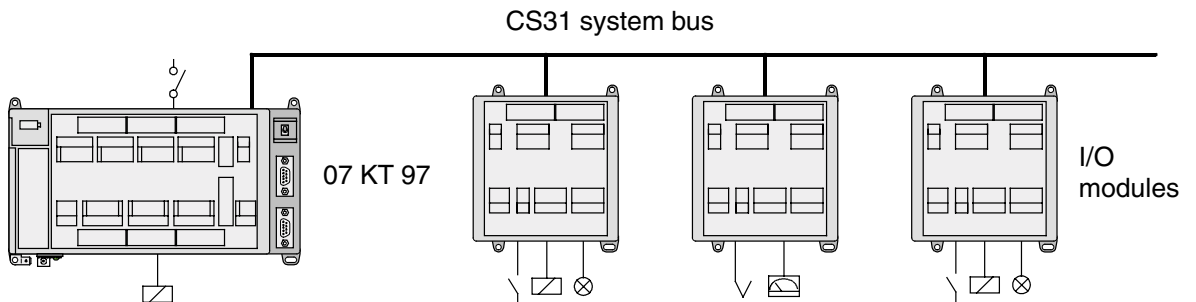
KW80,0 [ms] Monitoring of the task with a cycle time > 0	Cycle time violation	Infinite loop
0	Behaviour "like EBS"; the FK2 error is initiated after the cycle time is "violated" three times.	The watchdog initiates the FK2 error if the triple cycle time is exceeded.
>0	The task is paused after the PE for a period of 10 % of the set cycle time if the cycle time was violated three times. (During this time other PLC programs and operating system functions can be processed). Cycle time violations are entered to the following system flags: %MW1259,1 Task ID %MW1259,2 Error in ms %MW1259,3 Number of faulty cycles (cumulative counter for all PLC user tasks) The task returns to the set cycle time as soon as possible.	The watchdog initiates the FK2 error if the cycle time is exceeded by the value set in KW80,00 / %MW3080.0.
KW80,1 [ms] Monitoring of the task with a cycle time =0	Infinite loop	
0	The watchdog initiates the FK2 error after 10 s.	
>0	The watchdog initiates the FK2 error if the cycle time set in KW80,1 / %MW3080.1 is exceeded.	

5 Addressing of the 07 KT 97 on the CS31 system bus

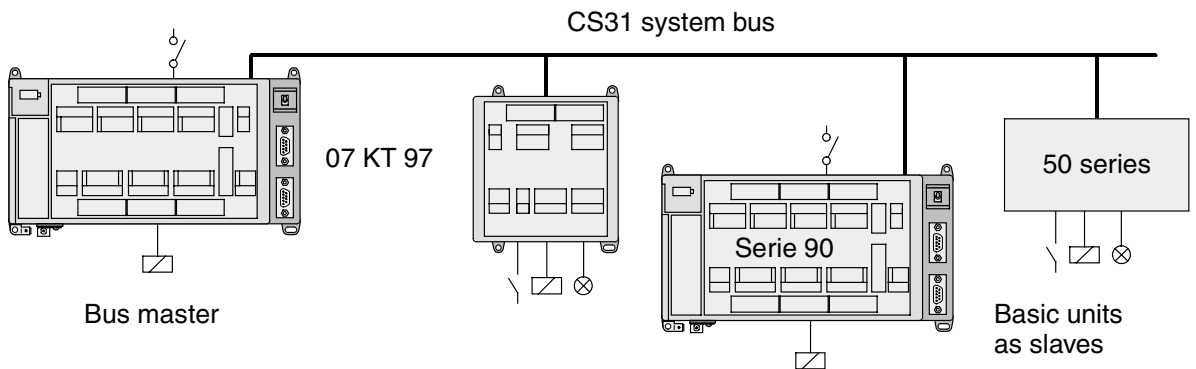
5.1 Introduction / Structure examples with 07 KT 97 as bus master



Example 1: 07 KT 97 as stand-alone PLC



Example 2: 07 KT 97 as bus master on the CS31 system bus, only I/O modules are used as remote modules



Example 3: 07 KT 97 as bus master and as slaves on the CS31 system bus, basic units of the series 50 and 90 as slaves, I/O modules

Regardless of the address ranges, the following modules can be connected to a CS31 system bus:

- max. 1 bus master
- max. 31 remote modules / slaves

Further restrictions result from the address range of the basic unit 07 KT 97:

- max. 28 analog input modules
- max. 28 analog output modules
- max. 31 digital input modules
- max. 31 digital output modules

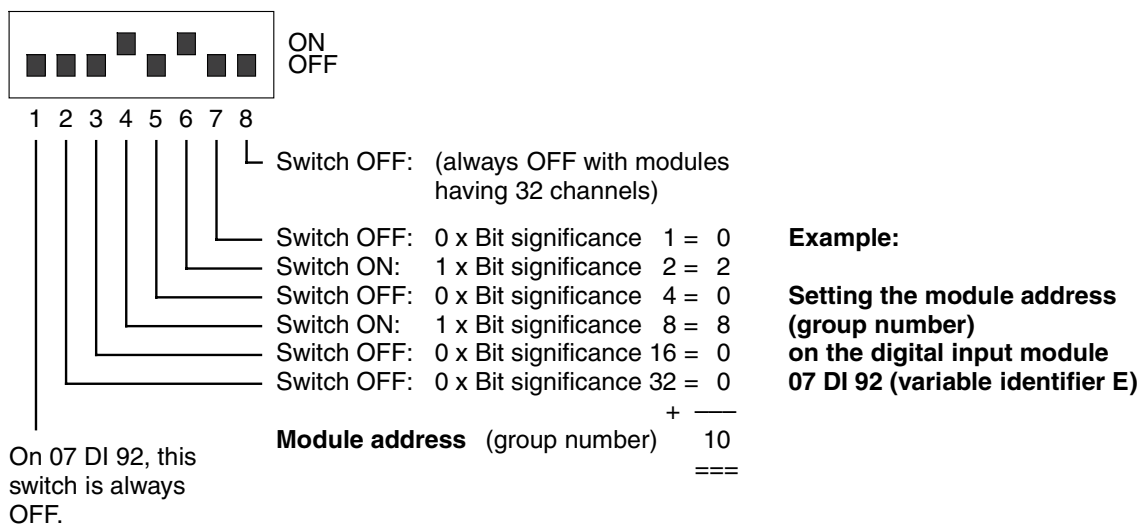
Depending on the structure of the installation and the types of the remote modules used, there may be further restrictions. For recommended addresses see chapter 1.4.2

Structure of the input and output addresses in the remote modules

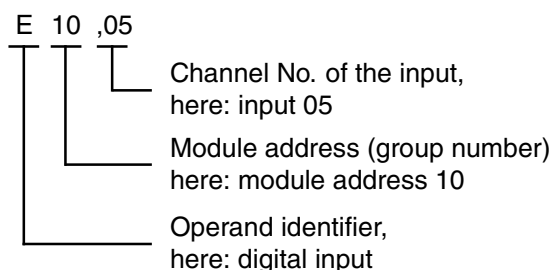
To enable the basic unit to correctly access the inputs and outputs, an address must be set for every module on the bus.

As an example, the binary input module **07 DI 92** is introduced here.

The address setting is done with the DIL switch located under the slide cover on the right hand side of the module housing (meaning of the 8 switches, see below).



The bus master basic unit reads the input signals as operands. The hardware identifier and the complete address of the input signal have the following structure:



In the programming system 907 AC 1131 the input declaration is performed as follows:

E10_05_switch1 AT %IX10.5 : BOOL; (* Switch 1 *)

Any symbolic identifier (here E10_05_switch1) and comment can be used.

The following table shows the input channel allocations which result from the use of the central unit 07 KT 97 as bus master.

Terminal	Input	Address	Terminal	Input	Address
5	E n,00	%lxn.00	30	E n+1,00	%lxn+1.00
6	E n,01	%lxn.01	31	E n+1,01	%lxn+1.01
7	E n,02	%lxn.02	32	E n+1,02	%lxn+1.02
8	E n,03	%lxn.03	33	E n+1,03	%lxn+1.03
9	E n,04	%lxn.04	34	E n+1,04	%lxn+1.04
10	E n,05	%lxn.05	35	E n+1,05	%lxn+1.05
11	E n,06	%lxn.06	36	E n+1,06	%lxn+1.06
12	E n,07	%lxn.07	37	E n+1,07	%lxn+1.07
15	E n,08	%lxn.08	40	E n+1,08	%lxn+1.08
16	E n,09	%lxn.09	41	E n+1,09	%lxn+1.09
17	E n,10	%lxn.10	42	E n+1,10	%lxn+1.10
18	E n,11	%lxn.11	43	E n+1,11	%lxn+1.11
19	E n,12	%lxn.12	44	E n+1,12	%lxn+1.12
20	E n,13	%lxn.13	45	E n+1,13	%lxn+1.13
21	E n,14	%lxn.14	46	E n+1,14	%lxn+1.14
22	E n,15	%lxn.15	47	E n+1,15	%lxn+1.15

07 DI 92: Addressen of the 32 input channels

n: Module address, can be set with the switches 2...7 on the address DIL switch.
 Recommended module addresses for the use of 07 KT 97 as bus master:
 08, 10, 12.....60 (even-numbered addresses)

The module seizes 2 addresses for inputs on the CS31 system bus.

The switches 1 and 8 of the address DIL switch must be set to OFF.



Note:

Some other modules may have a more or less different address setting. See the following chapters.

5.2 Recommended module addresses on the CS31 system bus

The standard addressing has the purpose of

- simplifying and schematizing the setting of addresses on the CS31 system bus and
- simplifying the diagnosis and troubleshooting.

The standard addressing makes sure that there will be no address overlappings even for modules with a higher amount of data.

Recommendation:

- Assign a specific module address for each module / each slave basic unit. That means the giving up of the possibility of a double assignment of module addresses by digital and analog modules.
- Module addresses 8, 10, ..., 58, 60 for digital remote modules and basic units (only even-numbered addresses). Refer also to chapter 1.4.6 "Intelligent I/O remote modules (basic units) as slave on the CS31 system bus".
- Module addresses 0...5 and 8...15 for analog remote modules.

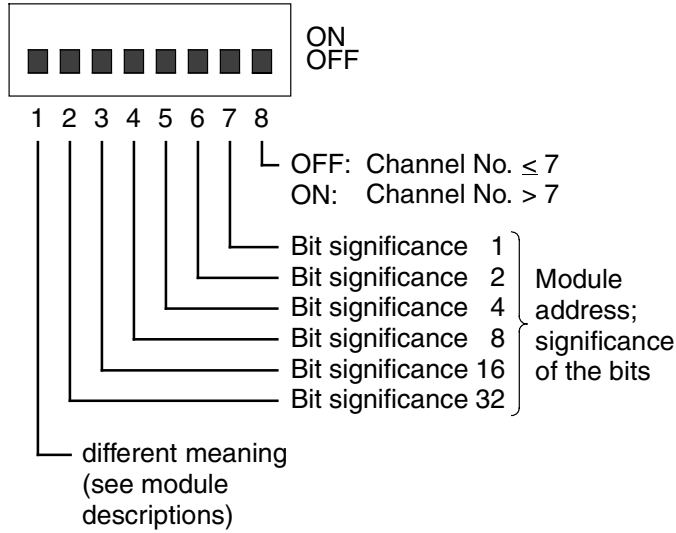
5.3 Address setting of the individual modules

Setting the address switch for digital modules

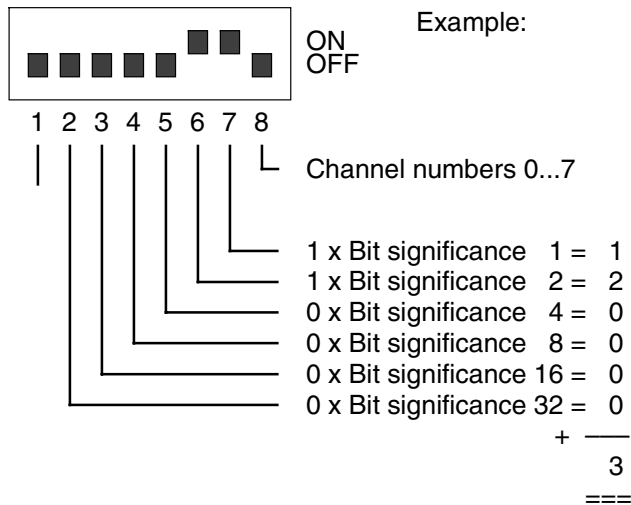
The possible module address range for the basic unit 07 KT 97 is:

0...61

The meaning of the individual switches is as follows:



Example for a digital modul: modul address 3



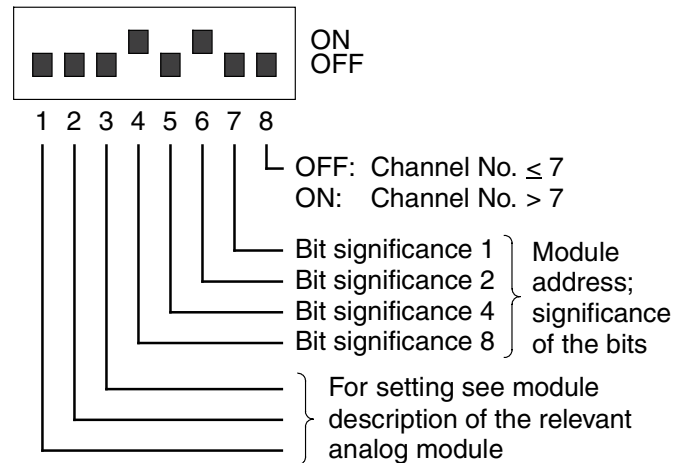
Setting the address switch for analog modules

When using the basic unit 07 KT 97 as bus master, the possible module address range for analog modules is:

0...5 and 8...15

The analog representation in the 07 KT 97 is fixed. The AC31 analog modules provide appropriate values. For more information concerning the analog representation, refer to the description of the analog modules "General information for the use of analog modules" in chapter "Analog modules" in volume 2.

The meaning of the individual switches is as follows:



5.4 07 KT 97 used as stand-alone basic unit

If the basic unit 07 KT 97 is to be used without the CS31 system bus connected, perform the following setting when programming the user program:

```
System constant KW 00,00 / %MW3000.0 := -2  
KW00_00_Mast_Slav AT %MW3000.0 : INT := -2; (* Stand-alone *)
```

This value is set on delivery.

5.5 07 KT 97 used as bus master basic unit

If remote modules (slaves) are connected to the basic unit 07 KT 97 via the CS31 system bus, proceed as follows:

1. Change the system constant: KW 00,00 = -1
KW00_00_Mast_Slav AT %MW3000.0 : INT := -1; (* Master *)
2. Save the PLC program in the Flash EPROM
3. Activate the new PLC mode by:
 - Selecting the menu item "Online/Reset" (warm start) in the ABB programming and test system
 - Selecting the menu item "Online/Cold start" in the ABB programming and test system
 - Power "ON"

5.6 Intelligent I/O-remote modules (basic units) as slave on the CS31 system bus

The basic unit 07 KT 97 can also be used as a slave on the CS31 system bus.

The basic unit 07 KT 97 may be used in the binary as well as in the word area.

The address values can range from 0 to 61 or 100 to 115. The maximum permissible address depends on the size of the set receive and transmit area. The larger you choose the receive or transmit area, the smaller is the maximum permissible address (see examples 1...4).

The slave basic unit may either be used in the digital area or in the word area on the CS31 system bus. If used in the word area the transmit or receive data are located on the channels 0...7 or 8...15, which is selected together with the address setting:

KW 00,00 / %MW3000.0 := 0...5; 8...15 → Channels 0...7

KW 00,00 / %MW3000.0 := 100...105; 108...115 → Channels 8...15

The upper channel range of 8...15 is configured by adding the value of 100 to the address.

In order to switch over to the "slave mode", proceed as follows:

1. Change the system constant: KW 00,00 / %MW3000.0 := 0...61
KW00_00_Mast_Slav AT %MW3000.0 : INT := 0...61; (* Slave *)
2. Save the user program in the Flash EPROM
3. Activate the new PLC mode by:
 - Selecting the menu item "Online/Reset" (warm start) in the ABB programming and test system
 - Selecting the menu item "Online/Cold start" in the ABB programming and test system
 - Power "ON"

There is no direct access to the inputs and outputs of the slave basic unit via the CS31 system bus. The communication between master and slave is performed using input and output operands.

All the master data are consistently transferred to the slave and all the slave data are consistently transferred to the master.

The slave basic unit may either be used in the binary area or in the word area on the CS31 system. Using the two system constants KW 00,10 / %MW3000.10 and KW 00,11 / %MW3000.11 the transmit and the receive area of the slave can be adapted to the application specific requirements (see also chapter "System constants").

It can be set:

- the size of the transmit and the receive area and
- whether the slave is to be used in the binary or in the word range.

Default condition:

If the basic units 07 KT 97 are switched over to the "slave mode", they behave on the CS31 system bus like input and output modules with 32 inputs and 32 outputs.

This means that the default setting of the transmit and receive area is within the binary area of the master and their size is 32 bits (4 bytes) each.

Example 1:

Default configuration of this slave (digital area):

KW00_10_SLV_SEND AT %MW3000.10 : INT := 0; (* 4 bytes – data exchange *)
 KW 00,10 = 0: Slave transmit area: 4 bytes (4 bytes * 8 channels = 32 digital outputs (A))
 KW00_11_SLV_REC AT %MW3000.11 : INT := 0; (* 4 bytes – data exchange *)
 KW 00,11 = 0: Slave receive area: 4 bytes (4 bytes * 8 channels = 32 digital inputs (E))



Note:

The default configuration is equal to the configuration
 KW 00,10 / %MW3000.10 = KW 00,11 / %MW3000.11 := 4

07 KT 97 as bus master Receiving or transmitting with I/O operands (E/A)	\longleftrightarrow	07 KT 97 as slave with: KW 00,10 / %MW3000.10 := 0 or 4 KW 00,11 / %MW3000.11 := 0 or 4 Transmitting or receiving with I/O operands (E/A)
E n ,00 / %IXn .0 : E n ,15 / %IXn .15 E n+1,00 / %IXn+1.15 : E n+1,15 / %IXn+1.15	\longleftarrow \longleftarrow	A 00,00 / %QX0.0 : A 00,15 / %QX0.15 A 01,00 / %QX1.0 : A 01,15 / %QX1.15
A n ,00 / %QXn .0 : A n ,15 / %QXn .15 A n+1,00 / %QXn+1.0 : A n+1,15 / %QXn+1.15	\longrightarrow \longrightarrow	E 00,00 / %IX0.0 : E 00,15 / %IX0.15 E 01,00 / %IX1.0 : E 01,15 / %IX1.15

n: Module address of the slave basic unit. For this example: $0 \leq n \leq 60$

For the slave address of n = 12 the following applies for example:

The output signal A 00,00 / %QX0.0 of the 07 KT 97 used as slave is the input signal E 12,00 / %IX12.0 for the 07 KT 97 used as bus master.


Example 2:

Configuration of the slave for the digital area:

KW00_10_SLV_SEND AT %MW3000.10 : INT := 15; (* 15 bytes – data exchange *)
 KW 00,10 = 15: Slave transmit area: 15 bytes (15 bytes * 8 channels = 120 digital outputs (A))

KW00_11_SLV_REC AT %MW3000.11 : INT := 6; (* 6 bytes – data exchange *)
 KW 00,11 = 6: Slave receive area: 6 bytes (6 bytes * 8 channels = 48 digital inputs (E))

07 KT 97 as bus master	←→	07 KT 97 as slave with: KW 00,10 / %MW3000.10 := 15 KW 00,11 / %MW3000.11 := 6 Transmitting or receiving with I/O operands (E/A)
Receiving or transmitting with I/O operands (E/A)		
E n ,00 / %IXn .0	←	A 00,00 / %QX0.0
⋮		⋮
E n ,15 / %IXn .15		A 00,15 / %QX0.15
⋮		⋮
E n+7,00 / %IXn+7.15		A 07,00 / %QX7.0
⋮		⋮
E n+7,07 / %IXn+7.7	←	A 07,07 / %QX7.7
⋮		⋮
A n ,00 / %QXn .0	→	E 00,00 / %IX0.0
⋮		⋮
A n ,15 / %QXn .15		E 00,15 / %IX0.15
⋮		⋮
A n+2,00 / %QXn+2.0		E 02,00 / %IX2.0
⋮		⋮
A n+2,15 / %QXn+2.15	→	E 02,15 / %IX2.15

 **Notes:**
 The upper 8 inputs channels of the address n+7
 E n+7,08 / %IXn+7.8...E n+7,15 / %IXn+7.15
 can be seized on the CS31 system bus with an other digital 8-bit input module (*excluding* KR/KT).
 The output channels beginning from the address n+3
 A n+3,00 / %QXn+3.00...A n+7,15 / %QXn+7.15
 can be seized on the CS31 system bus with an other digital 8-bit output module (*including* KR/KT).

n: Module address of the slave basic unit. For this example: $0 \leq n \leq 54$

For the slave address of n = 12 the following applies for example:

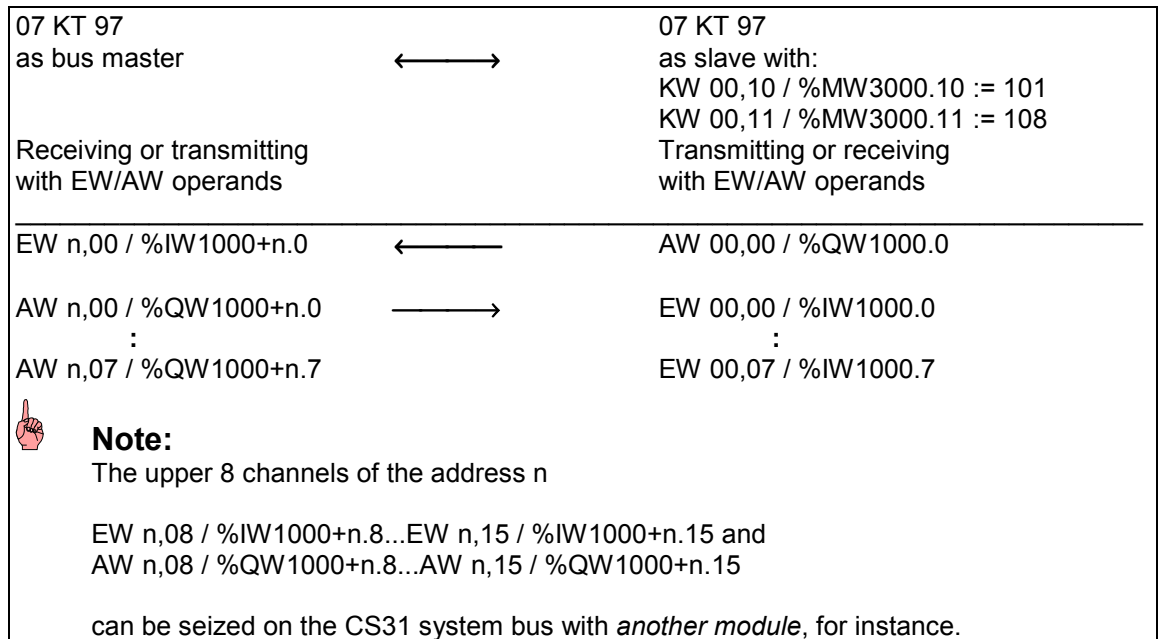
The output signal A 00,00 / %QX0.0 of the 07 KT 97 used as slave is the input signal E 12,00 / %IX12.0 for the 07 KT 97 used as bus master.

Example 3:

Configuration of the slave for the word area:

KW00_10_SLV_SEND AT %MW3000.10 : INT := 101; (* 1 word – data exchange *)
KW 00,10 = 101: Slave transmit area: 1 word (1 word = 1 word output)

KW00_11_SLV_REC AT %MW3000.11 : INT := 108; (* 8 words – data exchange *)
KW 00,11 = 108: Slave receive area: 8 words (8 words = 8 word inputs)



n: Module address of the slave basic unit. For this example: $0 \leq n \leq 5$ and $8 \leq n \leq 15$

For the slave address of $n = 4$ the following applies for example:

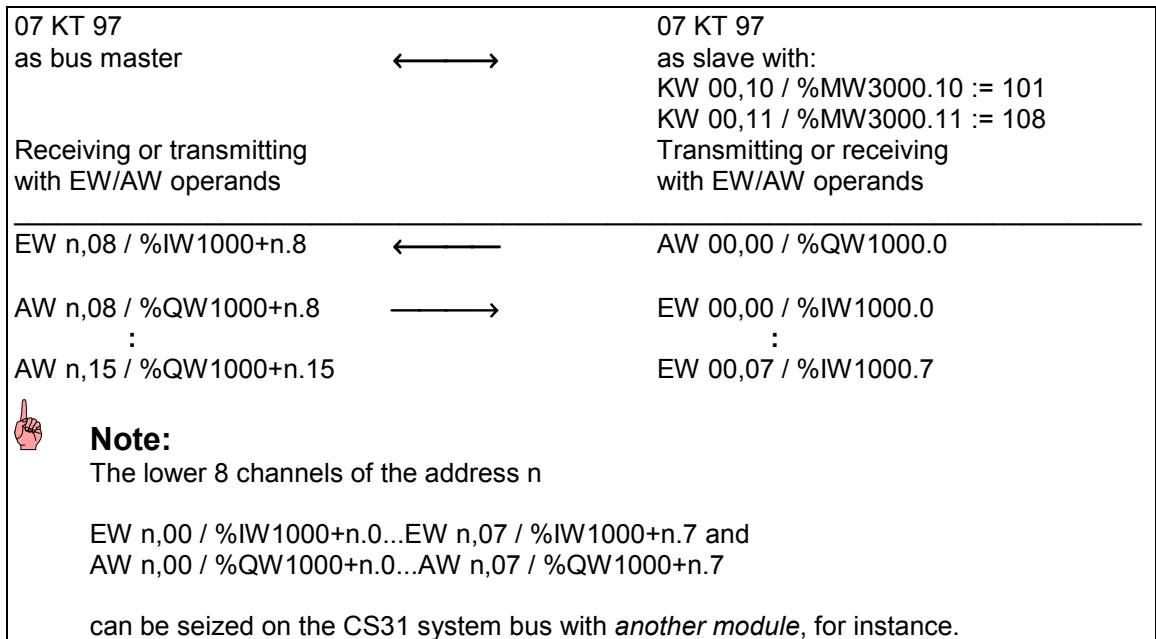
The output signal AW 00,00 / %QW1000.0 of the 07 KT 97 used as slave is the input signal EW 04,00 / %IW1004.0 for the 07 KT 97 used as bus master.

Example 4:

Configuration of the slave for the word area:

KW00_10_SLV_SEND AT %MW3000.10 : INT := 101; (* 1 word – data exchange *)
 KW 00,10 = 101: Slave transmit area: 1 word (1 word = 1 word output)

KW00_11_SLV_REC AT %MW3000.11 : INT := 108; (* 8 words – data exchange *)
 KW 00,11 = 108: Slave receive area: 8 words (8 words = 8 word inputs)



n: Module address of the slave basic unit. For this example: $100 \leq n \leq 105$ and $108 \leq n \leq 115$

For the slave address of $n = 104$ the following applies for example:

The output signal AW 00,00 / %QW1000.0 of the 07 KT 97 used as slave is the input signal EW 04,08 / %IW1004.8 for the 07 KT 97 used as bus master.

5.7 Special modules used as slave on the CS31 system bus

Festo valve island / installation island

The Festo valve island and the Festo installation island behave on the CS31 system bus like binary input and output modules. Refer to the Festo documentation for the scope of seized data.

5.8 Complex structure examples including addresses

Categorization of the modules with respect to the I/O terminals:

There are two main module types:

- **digital modules:** These modules are controlled by means of binary I/O operands (E or A). The basic units 07 KT 97 also belong to this type, if they are used as slaves.
- **analog modules:** These modules are controlled by means of word I/O operands (EW or AW). The basic unit 07 KT 97 also belong to this type as well as the high-speed counter ICSF 08 D1 which receives its preset data as word data, for example.

The following table shows an overview of the module types. These designations will be used in example 6.

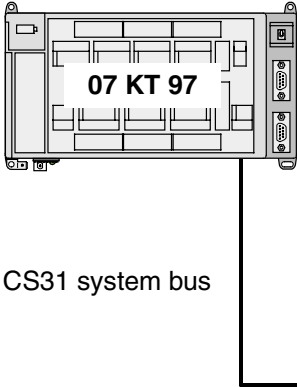
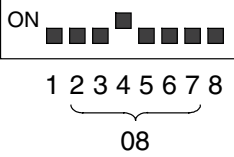
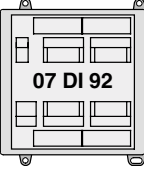
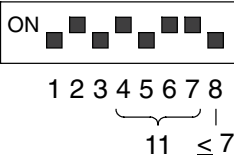
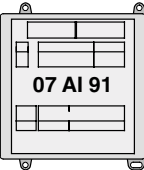
Please note, that the configurable binary modules 07 DC 91 and 07 DC 92 behave differently according to the performed setting.

5.9 Module examples (slaves on the CS31 system bus)

Module types, with regard to the I/O terminals	Module examples
Digital output modules with 32 outputs	07 DC 92 with switch 1 is set to ON
Digital modules with 16 inputs and 16 outputs	07 DC 91 with switch 1 is set to ON
Digital modules with 32 inputs and 32 outputs	07 DC 92 with switch 1 is set to OFF
Digital modules with 120 signals from and 120 signals to the CS31 system bus	07 KT 9x, 07 KR 51, 07 KT 51 as slave
Analog input modules with 8 inputs	07 AI 91
Analog output modules with 8 outputs	07 AC 91 configured as an output module
Analog modules (word modules) with up to 8 inputs and 8 outputs	07 KT 9x, 07 KR 51, 07 KT 51, 07 AC 91

Examples for the assignment of module address

Example 5:

 <p>CS31 system bus</p>		<p>07 KT 97</p> <p>E 62,00 / %IX62.0 : E 62,15 / %IX62.15 E 63,00 / %IX63.0 : E 63,15 / %IX63.15 A 62,00 / %QX62.0 : A 62,15 / %QX62.15 A 63,00 / %QX63.0 : A 63,07 / %QX63.7 EW 06,00 / %IW1006.0 : EW 06,07 / %IW1006.7 AW 06,00 / %QW1006.0 : AW 06,03 / %QW1006.3</p>	<p>Addresses of the inputs and outputs on the bus master basic units</p>	<p>Bus master basic unit</p>
<p>Address switch (DIL switch) on the module</p>	<p>Remote module</p>	<p>Address in the program of the master CPU</p>		
 <p>ON</p> <p>1 2 3 4 5 6 7 8</p> <p>08</p>	 <p>07 DI 92</p> <p>32 digital inputs</p>	<p>E 08,00 / %IX8.0 1st digital input : E 08,15 / %IX8.15 : E 09,00 / %IX9.0 : E 09,15 / %IX9.15 32nd digital input</p>	<p>Remote modules (slaves)</p>	
 <p>ON</p> <p>1 2 3 4 5 6 7 8</p> <p>11 ≤ 7</p>	 <p>07 AI 91</p> <p>8 analog inputs</p>	<p>EW 11,00 / %IW1011.0 1st analog input EW 11,01 / %IW1011.1 2nd analog input : EW 11,07 / %IW1011.7 8th analog input</p>		

Example 6:

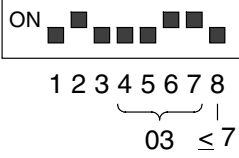

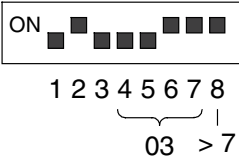



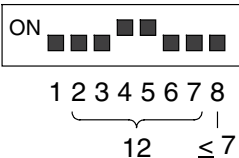

See next page

07 KT 97

Address setting in the master:
KW 00,00 / %MW3000.0 = -1

E 62,00...E 62,15
E 63,00...E 63,15
A 62,00...A 62,15
A 63,00...A 63,07
EW 6,00...EW 6,07
AW 6,00...AW 6,03

Inputs /
outputs
on the
basic unit

Address switch on the remote module	Remote module	Address in the program of the master CPU	Remarks
			- max. 31 slaves on the CS31 bus
 <p>ON  1 2 3 4 5 6 7 8 03 ≤ 7</p>	07 AI 91 8 inputs analog E0 : E7	EW 03,00 %IW1003.0 : EW 03,07 %IW1003.7	- Permissible range of module addresses for analog modules: 0...5 and 8...15
 <p>ON  1 2 3 4 5 6 7 8 03 > 7</p>	07 AI 91 8 inputs analog E0 : E7	EW 03,08 %IW1003.8 : EW 03,15 %IW1003.15	- 2 analog input modules with 8 channels each can be assigned to one address (16 channels together).
 <p>ON  1 2 3 4 5 6 7 8 09 ≤ 7</p>	07 AC 91 8 outputs analog 8 inputs analog A0 : A7 E0 : E7	AW 09,00 %QW1009.0 : AW 09,07 %QW1009.7 EW 09,00 %IW1009.0 : EW 09,07 %IW1009.7	- The same address (as for the analog input modules) may also be used for the connection of analog output modules (as shown to the left).
 <p>ON  1 2 3 4 5 6 7 8 12 ≤ 7</p>	07 DI 92 32 inputs digital E00 : E31	E 12,00 %IX12.0 : E 12,15 %IX12.15 E 13,00 %IX13.0 : E 13,15 %IX13.15	<ul style="list-style-type: none"> - Permissible range of module addresses: for digital modules: 0...61, recommendation: 6...60 - The following might be done, but does not bring you any advantage: <ul style="list-style-type: none"> • using the same addresses for digital modules as for analog modules • collecting input and output under one address
<div style="border: 1px solid black; padding: 5px; width: fit-content;"> 07 KT 97: KW 0,0 / %MW3000.0 = 4 </div>	up to 120 digital inputs and up to 120 digital outputs or up to 8 word inputs and up to 8 word outputs	KR/KT in the digital range: E 04,00...E 11,07 / %IX4.0...%IX11.7 A 04,00...A 11,07 / %QX4.0...%QX11.7 or KR/KT in the word range: EW 04,00...EW 04,07 / %IW1004.0...%IW1004.7 AW 04,00...AW 04,07 / %QW1004.0...%QW1004.7	<ul style="list-style-type: none"> - Slave-KR/KT with 120 E and/or 120 A occupies the set and the following 7 addresses (only half of the 7th, though). For address 4 of the example: <ul style="list-style-type: none"> • Next free address for KT: Bit range: 12, word range: 5 • Max settable KT address: Bit range: 54, word range: 5

6.1 Purpose of the I/O configuration at I/O modules

Depending on the type of I/O module the following can be configured:

- For digital modules with combined I/O channels, these channels can be configured as "only inputs" or "only outputs".
- For analog modules, measuring ranges or output areas can be configured which are different from the factory setting.

Switching over of inputs and outputs, switching on the diagnosis functions and changing the measuring ranges and output areas is performed as follows, depending on the module type:

- Performing the I/O configuration via the CS31 system bus by means of the user program of the bus master basic unit.
- Setting of switches on the remote module.
- External wiring of the input/output module.

In some cases there is a relation between the settings made on the remote module and the information and diagnosis messages which can be interrogated at the remote module via the CS31 system bus. This relation is explained in the following chapters.

Do not perform an I/O configuration via the CS31 system bus if the factory setting is sufficient. Once an I/O configuration has been performed, it will remain stored in the corresponding I/O module until it is changed again by means of an I/O configuration process. Even in case of power OFF it will not be deleted.

6.2 Performing and reading the I/O configuration:

There are the following possibilities for system structures with 07 KT 97 as bus master:

- Performing and reading the I/O configuration via the user program of the bus master basic unit 07 KT 97.
- Reading the I/O configuration from the remote modules.

6.2.1 Performing and reading the I/O configuration via the user program

The function block CS31CO is available for the I/O configuration of the modules. This function block is part of the programming software 907 AC 1131 and is described in the corresponding documentation.

6.2.2 Reading of I/O configuration and diagnosis data at the remote module

Reading the I/O configuration and the diagnosis information for an I/O terminal of a module will be shown in the following for the module ICSC 08 L1 as an example. The procedure is the same for all remote modules, only the type and the amount of diagnosis information are different. To read the information, the test button (4) and the LED displays (1) of the module have to be used.

When the test button is pressed for the first time the channel E/A0 (input/output 0) is selected: LED0 flashes. After releasing the button, the LEDs 0 to 7 display the diagnosis information of this channel for approx. 3 seconds.

The LEDs have the following meaning:

- 0 UE = Unit error
- 1 BE = Bus error
- 2 unused
- 3 CI/CO = Cut wire of inputs/outputs
- 4 OL = Overload
- 5 SC = Short circuit
- 6 Configuration as an output
- 7 Configuration as an input. If the LEDs 6 and 7 light up simultaneously, the channel is configured as a combined input/output.

The meaning of the LEDs (2) is also printed in English on the front panel of the module.

The operation is repeated for the other channels each time the test button is pressed and released again.

After the last channel E/A7 (input/output 7) has been scanned, pressing the test button again causes a lamp test (LED test) to be performed. All 8 LEDs should light up. After releasing the button, the LEDs display the settings of the DIL switch on the plug-in base for approx. 5 seconds. LED 0 displays the setting of switch 1 (LEDs 0...7 are assigned to the switches 1...8).

All error messages are stored in the module and can only be deleted by pressing the test button for 10 seconds or by switching the power OFF and ON.

7.1 Introduction

The diagnosis system of the 07 KT 97 is designed to ensure a quick and efficient troubleshooting. For this purpose the diagnosis system is organized as follows:

- "Vertically" in diagnosis, error flags, reactions, LED displays and acknowledgement. There are interrelations between the bus master basic unit and the remote modules: The basic unit reads the diagnosis data detected by the remote modules. An acknowledgement in the basic unit causes also the deletion of the error messages stored in the remote modules.
- "Horizontally" in 4 error classes, corresponding to the severity of the errors.

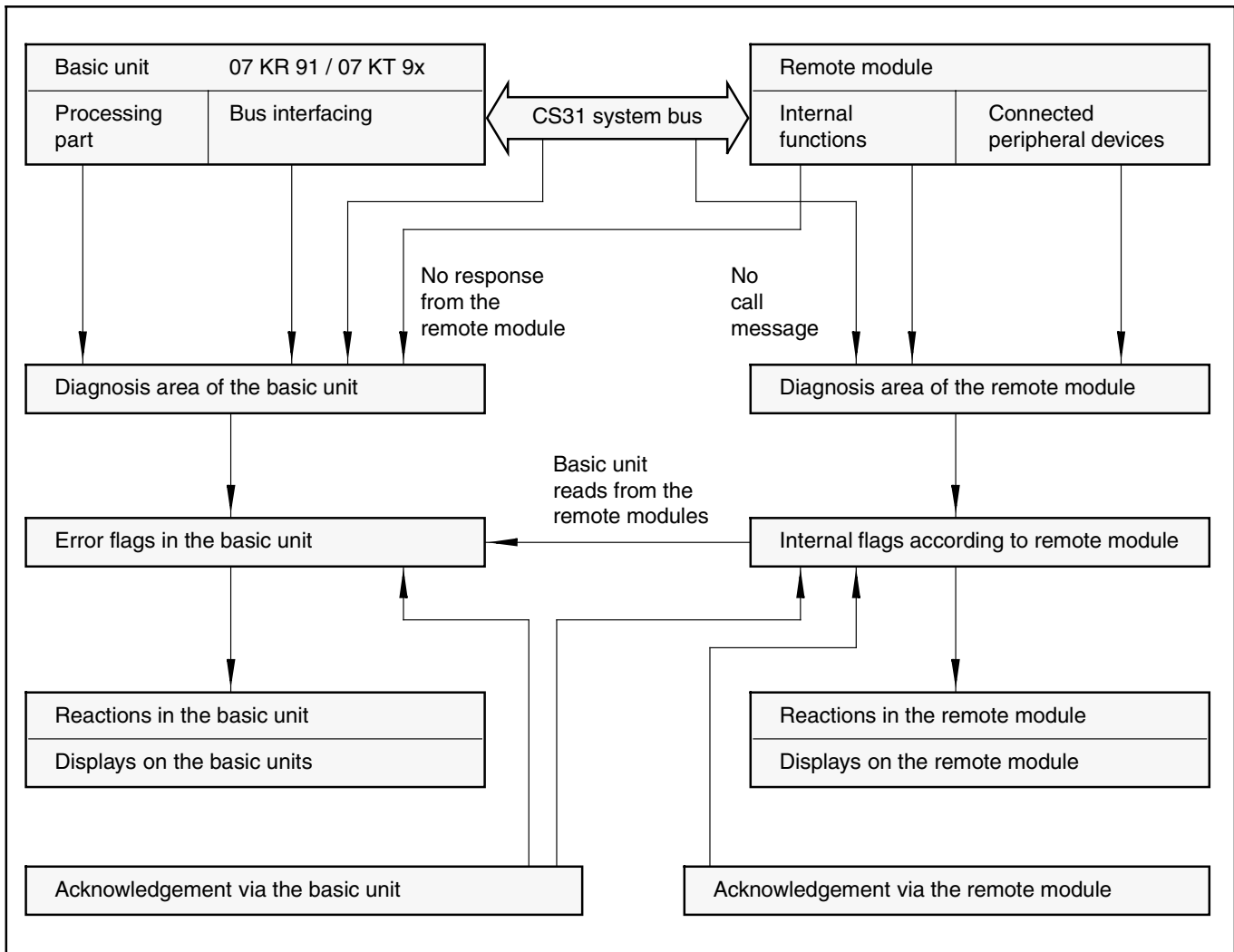
This concept is based on a system structure consisting of a bus master basic unit and several remote modules, and remote processors as well. The diagnosis system detects the following errors:

- Errors in the bus master basic unit
- Errors on the CS31 system bus
- Errors in the remote modules
- Errors in the wiring of the remote modules on the process side

The troubleshooting is performed as follows:

- The LEDs on the basic unit 07 KT 97 give first hints. The errors detected by the remote modules are also displayed here.
- If these hints are not sufficient, the error flags have to be read out.
- The status register EW 07,15 / %IW1007.15 in the basic unit supplies additional information to be used for the diagnosis.
- The remote modules indicate errors occurring in their area. Detailed information can be obtained by pressing the test button on the modules.

7.2 Structure of the diagnosis



7.3 Troubleshooting by means of the LED displays on the basic unit

The LED displays on the front panel of the basic unit supply initial information on the errors which occurred:

- BA= CS31 bus processor active
- BE= Bus Error (error on the CS31 system bus)
- RE= Remote Unit Error (error in/on a remote module)
- SE= Serial Unit Error (error in the CS31 bus interfacing of the basic unit)
- RUN = User program is running (no error)
- FK1 = Error class 1 (fatal error)
- FK2 = Error class 2 (serious error)
- FK3 = Error class 3 (light error)
- Supply = Power supply available
- Battery = Battery available / User program is being flashed
- Ovl = Overload or short circuit on at least one direct digital output of the basic unit

If no LED lights up, the basic unit has not found any error. Exception: The LED Battery (battery is missing); the battery is only necessary for certain cases of application.

The LED Battery flashes during the process of writing the user program into the Flash memory.

7.3.1 LEDs for CS31 system bus and bus interfacing

LED	BA BE RE SE	Meaning	Remedy
Basic unit	gn rd rd rd		
A M S	* ○ ○ ○	Everything is OK.	--
A M S	○ ⊙ ⊙ ⊙	A fatal error occurred. The watchdog has switched off the CS31 bus. All outputs are OFF (0 state).	<ul style="list-style-type: none"> • Power OFF/ON. If no success, module is defective. • Evaluate the error flags.
A M S	* ○ ○ *	Dual-port RAM defective	<ul style="list-style-type: none"> • Power OFF/ON. If no success, module is defective. • Evaluate the error flags.
A M S	○ * * *	Initialization phase after power ON or after cold start.	--
M	* * ○ ○	Master basic unit does not find any remote modules on the CS31 bus <u>after</u> power ON or after cold start.	<ul style="list-style-type: none"> • Install remote modules. • Check the CS31 bus line. • Check the power supply of the remote modules. • Evaluate the error flags.
M	* ○ * ○	Error message from a remote module.	<ul style="list-style-type: none"> • Evaluate the error flags. • Check the remote modules.
M	* * * ○	One remote module can suddenly not be controlled by the master basic unit any more.	<ul style="list-style-type: none"> • Evaluate the error flags. • Check the power supply of the remote modules. • Check the CS31 bus line. • Check the remote module.
M	* * ○ ○	There are at least 3 remote modules on the CS31 bus. Two of the remote modules can suddenly not be controlled by the master basic unit any more.	<ul style="list-style-type: none"> • Evaluate the error flags. • Check the power supply of the remote modules. • Check the CS31 bus line. • Check the remote module.
M	* * * *	There are at least 2 remote modules on the CS31 bus. Suddenly no remote module can be controlled by the master basic unit any more.	<ul style="list-style-type: none"> • Evaluate the error flags. • Check the power supply of the remote modules. • Check the CS31 bus line. • Check the remote module.
S	* ○ ⌘ ○	CS31 bus does not work.	<ul style="list-style-type: none"> • Check the CS31 bus line. • Check the master basic unit.

○ = LED off, * = LED on, ⌘ = LED flashes, ⊙ = LED on or off, gn = green, rd = red,
A = Stand-alone master or slave basic unit, M = master basic unit, S = slave basic unit

7.3.2 LEDs for user program and error display

LED	R F F F	Meaning	Remedy
Basic unit	U K K K N 1 2 3 gn rd rd rd		
A M S	* O O O	User program is running.	--
A M S	* O O *	User program is running, but a light error occurred.	• Evaluate the error flags and eliminate the error.
A M S	O O O O	User program does not run.	• Start the user program.
A M S	O O O *	A light error occurred which caused the user program to be aborted automatically because the system constant KW00,07 is not equal to "0".	• Evaluate the error flags and eliminate the error.
A M S	O O * O	A serious error occurred which caused the user program to be aborted automatically.	• Evaluate the error flags and eliminate the error, if possible.
A M S	O * O O	A fatal error occurred. The user program cannot be started.	• Evaluate the error flags. • Power OFF/ON. If no success, module is defective.
A M S	O O * *	A light and a serious error occurred.	• Evaluate the error flags and eliminate the error, if possible.
A M S	⊙ ⌘ ⊙ ⊙	Power-fail	• Power OFF/ON
A M S	* * * *	Initialization phase, power ON, cold start, warm start.	--
O = LED off, * = LED on, ⌘ = LED flashes, ⊙ = LED on or off, gn = green, rd = red, A = Stand-alone master or slave basic unit, M = master basic unit, S = slave basic unit			

7.3.3 LEDs for supply voltage and battery

LED	Supply Battery	Meaning	Remedy
Basic unit	gn rd		
A M S	* O	Supply voltage and battery available.	--
A M S	* *	Supply voltage available, battery not available.	--
A M S	* ⌘	Supply voltage available, the user program is being loaded into the Flash memory.	--
A M S	O O	Supply voltage is not available.	• Switch power ON. • Check the supply voltage.
O = LED off, * = LED on, ⌘ = LED flashes, ⊙ = LED on or off, gn = green, rd = red, A = Stand-alone master or slave basic unit, M = master basic unit, S = slave basic unit			

7.3.4 LEDs for overload or short-circuit on at least one direct digital output

LED	Ovl	Meaning	Remedy
Basic unit	Rd		
A M S	*	Overload or short-circuit on at least one one of the direct digital outputs.	• Eliminate the overload or short-circuit.
A M S	O	No overload or short-circuit	--
O = LED off, * = LED on, ⌘ = LED flashes, ⊙ = LED on or off, gn = green, rd = red, A = Stand-alone master or slave basic unit, M = master basic unit, S = slave basic unit			

7.4 Troubleshooting on remote modules

7.4.1 Diagnosis displays on the remote modules

Error LED:

The error LED contains the following information:

- OFF - CS31 bus is running (no error)
- no peripheral error
- Flashes - CS31 bus does not run
(the BE LED on the master is switched on),
i.e. no or permanently interfered communication with the master.
Causes may be:
 - The CS31 system bus is broken, short-circuited or wired in reversed polarity.
 - The basic unit is no longer configured as the bus master,
see also the system constant KW00_00 / %MW3000.0.
- ON - Module has detected a peripheral error
(the RE LED on the master is switched on)

Detailed information on the peripheral error:

If the remote module has detected an error, the error LED lights up.

The remote module supplies detailed error information via the 8 LEDs if the test button is pressed; see also the module descriptions.

The procedure will be explained in the following for the module 07 DC 92.

When the test button is pressed for the first time the channel E/A0 (input/output 0) is selected: LED0 flashes. After releasing the button, the LEDs 0 to 7 display the diagnosis information of this channel for approx. 3 seconds.

The LEDs have the following meaning:

0	Unused
1	Unused
2	Unused
3	Unused
4	OL = Overload
5	Unused
6	Unused
7	Unused

The meaning of the LEDs is also printed in English on the front panel of the module. The operation is repeated for the other channels each time the test button is pressed and released again.

After the last channel has been scanned, pressing the test button again causes a lamp test (LED test) to be performed. All LEDs must light up. After releasing the button, the LEDs display the settings of the DIL switch on the plug-in base for approx. 3 seconds. LED 0 displays the setting of switch 1 (LEDs 0...7 are assigned to the switches 1...8).

All error messages are stored in the module and can only be deleted by pressing the test button for 10 seconds or by switching the power OFF and ON.

Self test:

During the initialization phase, a self test is performed in the remote module. With a negative result, the module lights ON the error LED and **does not contact** the CS31 bus.

7.4.2 Diagnostic messages of the remote modules to the CS31 bus master

The diagnostic messages which are transferred to the master via the CS31 bus, are available in the system flags.

All the diagnostic messages are entered as FK4 errors (warnings) into the following system flags:

M255_10 / %MX255.10 := TRUE	Sum error message
M255_14 / %MX255.14 := TRUE	FK4 = warning
MW255_08 / %MW1255.8	Error number
MW255_09 / %MW1255.9	Device type
MW255_10 / %MW1255.10	Group number
MW255_11 / %MW1255.11	Channel number

The following table shows the reported messages for the different I/O modules.

Module name	Error number	Device type	Remarks
	%MW1255.8	%MW1255.9	
07 DI 92	-	-	-
07 DC 91	4	4	Overload/short-circuit at an output
07 DC 92	4	4	Overload/short-circuit at an output
07 AI 91	9	1	Cut wire
	10	1	Out of range
07 AC 91	10	1 / 5 *)	Analog input out of range
07 DI 93	4	0	Overload of input supply
07 DO 93	4	2	Overload at an output
07 DK 93	4	4	Overload of input supply or at an output

*) 1 - if only inputs are configured
5 - if inputs and outputs are configured

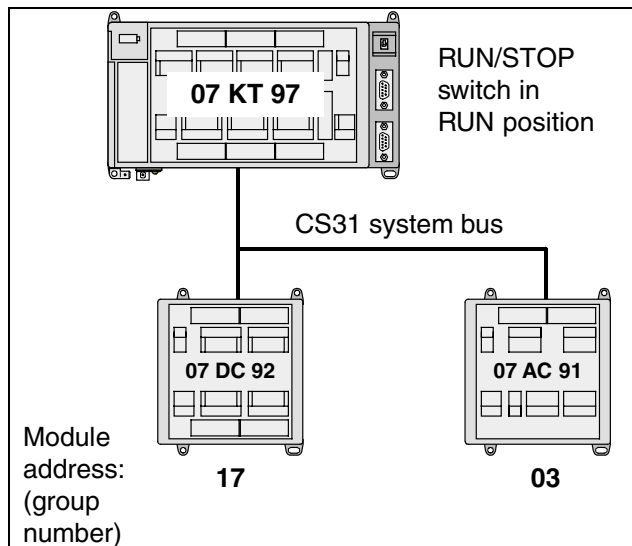
7.5 Acknowledgement of error messages in the remote modules

The remote modules store and display the error messages occurring there, independently of the basic unit. The error messages can be acknowledged:

- on the remote module by pressing the test button
- in the user program with the function block CS31QU (This also clears the error message stored in the basic unit)

If the error has not been eliminated, the error message appears again.

7.5.1 Example for an error message



Errors which occurred:

The bus line to the remote module with the module address 3 has been broken during operation.

Error flags in the basic unit 07 KT 97:

It is assumed that the error flags have been set to 0 by acknowledgement/deletion before the error occurred. In the following only those error flags are listed the contents of which changes.

- | | |
|-------------------------------|---|
| • M 255,10 / %MX255.10 = TRUE | Sum error message |
| • M 255,13 / %MX255.13 = TRUE | Error class message (FK3 error) |
| • MW 255,00 / %MW1255.0 = 15 | Error identification: remote module is disconnected |
| • MW 255,01 / %MW1255.1 = 05 | Module type: analog input and output |
| • MW 255,02 / %MW1255.2 = 03 | Group number (module address) |
| • MW 255,03 / %MW1255.3 = 0 | } not concerned, as well as
} all other error flags which
} have not been mentioned |
| • MW 255,04 / %MW1255.4 = 0 | |
| • MW 255,05 / %MW1255.5 = 0 | |
| • MW 255,06 / %MW1255.6 = 0 | |
| • MW 255,07 / %MW1255.7 = 0 | |
| | |
| | |

7.5.2 LED displays on the bus master basic unit 07 KT 97

- BA lights up ⇒ CS31 bus processor is active. The data exchange with the remote module having the address 17 is continued.
- BE lights up ⇒ Bus Error, error on the CS31 system bus.
- RE lights up ⇒ Remote Unit Error, error on a remote module.
- SE off ⇒ Serial Unit Error, the CS31 bus interfacing in the basic unit works correctly.
- RUN lights up
- FK3 lights up ⇒ light error

7.5.3 Reaction of the bus master basic unit 07 KT 97

The processing program and the bus operation continue running (if KW 00,07 / %MW3000.7 := 0).

Reaction of the remote module 07 AC 91: All outputs turn to OFF (0 state).

Reaction of the remote module 07 DC 92: Data exchange with the bus master basic unit 07 KT 97.

7.5.4 Status word EW 07,15 in the basic unit 07 KT 97

- Bit 0 = 1 No class 2 error
- Bit 1 = 0 Only applicable for 07 KT 97 as slave
- Bit 2 = 1 Date and time are valid
- Bit 3 = 1 Battery available
- Bit 4...7 Unused
- Bit 8...15 = x Max. number of modules connected to the CS31 system bus which have been found since the last power-on operation. The number will not be changed by the error which has occurred in the meantime.

7.5.5 Acknowledgement of the error flags in the basic unit 07 KT 97

Eliminate the error before acknowledgement. Otherwise the error message will appear again.

The bit flags M 255,10 / %MX255.10 and M 255,13 / %MX255.13 can be acknowledged by the following:

- Power ON
- Cold start (menu item in 907 AC 1131)
- Setting the RUN/STOP switch to RUN
- Overwriting the flag M 255,13 / %MX255.13 with "FALSE" (0) in the user program
- Using the online function "Overwrite" in order to overwrite the flag M 255,13 / %MX255.13 with "FALSE" (0)
- Using the function block CS31QU in the user program. (The function block CS31QU is applicable only for errors which concern the CS31 system bus. It also deletes the error message in the remote module.)

The relevant LEDs turn off upon the acknowledgement.

The word flags MW 255,00...MW 255,07 / %MW1255.0...%MW1255.7 can only be deleted by overwriting them. They are overwritten by newly occurring errors.

7.5.6 Acknowledgement of the error flags in the remote modules

- On the remote module by pressing the test button for a longer time.
- In the user program of the basic unit using the block CS31QU.

7.6 Error flags in the basic unit, error classification

The basic unit offers error messages for the user program which are classified into 4 error classes (FK1...FK4) according to their severity. The error messages are stored in error flags and can be used in the user program and be read by the programming system.

The following table gives an overview of the error flags:

Error class	FK1 = Fatal error	FK2 = Serious error	FK3 = Light error	FK4 = Warning
General feature of the error class, examples	Save operation of the operating system is no longer ensured. <u>Error examples:</u> - Checksum error in the operating system EPROM - Write/read error while testing the operating system RAM	The operating system works correctly, but the error-free processing of the user program is not ensured. <u>Error examples:</u> - Checksum error in the operating system Flash EPROM - Write/read error while testing the user program RAM	The choice whether the user program has to be aborted by the operating system or not depends on the application. The user decides which reactions are to be initiated. <u>Error examples:</u> - The Flash EPROM cannot be programmed - The dual port RAM to the CS31 part (LED SE) is faulty - The remote module has failed (LED RE)	Errors which occur on peripheral devices or which will have an effect only in the future. The user decides which reactions are to be initiated. <u>Error examples:</u> - Short-circuit on a remote module
Sum error message ¹⁾	M 255,10 / %MX255.10			
Error class message (if 1, an error exists)	M 255,11 / %MX255.11	M 255,12 / %MX255.12	M 255,13 / %MX255.13	M 255,14 / %MX255.14
Fehlerkennung (Wort) ²⁾ Detailinfo 1 (Wort) ²⁾ Detailinfo 2 (Wort) ²⁾ Detailinfo 3 (Wort) ²⁾ Detailinfo 4 (Wort) ²⁾ Detailinfo 5 (Wort) ²⁾ Detailinfo 6 (Wort) ²⁾ Detailinfo 7 (Wort) ²⁾	MW 254,00/%MW1254.0 MW 254,01/%MW1254.1 MW 254,02/%MW1254.2 MW 254,03/%MW1254.3 MW 254,04/%MW1254.4 MW 254,05/%MW1254.5 MW 254,06/%MW1254.6 MW 254,07/%MW1254.7	MW 254,08/%MW1254.8 MW 254,09/%MW1254.9 MW 254,10/%MW1254.10 MW 254,11/%MW1254.11 MW 254,12/%MW1254.12 MW 254,13/%MW1254.13 MW 254,14/%MW1254.14 MW 254,15/%MW1254.15	MW 255,00/%MW1255.0 MW 255,01/%MW1255.1 MW 255,02/%MW1255.2 MW 255,03/%MW1255.3 MW 255,04/%MW1255.4 MW 255,05/%MW1255.5 MW 255,06/%MW1255.6 MW 255,07/%MW1255.7	MW255,08/%MW1255.8 MW255,09/%MW1255.9 MW255,10/%MW1255.10 MW255,11/%MW1255.11 MW255,12/%MW1255.12 MW255,13/%MW1255.13 MW255,14/%MW1255.14 MW255,15/%MW1255.15
LED displays after initialization	FK1 lights up or LED RUN does not light up when the RUN/STOP switch is set to RUN	FK2 lights up or LED RUN does not light up when the RUN/STOP switch is set to RUN	FK3 lights up. In addition according to error type: LED BE (Bus Error) LED RE (Remote Unit Error) LED SE (Serial Unit Error)	LED RE (Remote Unit Error) lights up
Reaction when switching on the basic unit / Reaction during operation	All outputs remain FALSE or are set to FALSE. The programming system does not have access. Caution: Both processors of the basic unit monitor each other mutually, thus facilitating a powerful diagnosis. If the safety requirements are higher, use specially approved control systems.	All outputs remain set to FALSE or are set to FALSE. The programming system can get access. The user program is not started or is aborted.	It can be chosen in case of an error: - Just report the error: Evaluate the error flag M 255,13/%MX255.13 - Abort the user program: Set system constant KW 0,7/%MW1000.7 = 1 (FK3_REAK)	Evaluation of the error messages using the user program
Acknowledgement of the sum error message or the error class message	- Power ON - Cold start	- Power ON - Cold start	- Power ON / Cold start - RUN/STOP switch to RUN - Start program using 907 AC 1131 - M255,13 / %MX255.13 or M255,14 / %MX255.14 to FALSE - In case of CS31 error: function block CS31QU	
¹⁾ The sum error flag M 255,10 / %MX255.10 becomes TRUE, if at least one of the error class flags becomes 1. If M 255,10 / %MX255.10 = FALSE the basic unit has not detected any error. The sum error flag is automatically deleted when the error class flags are acknowledged.		²⁾ The basic unit enters the most recent found error into the relevant word flag set for each error class. The entry is made at the end of the program cycle and remains unchanged during the next running program cycle. The word flags can only be acknowledged by overwriting them with "0".		

7.7 Acknowledgement of error messages in the basic

Error messages remain stored and are displayed until they are acknowledged. The following applies:

- The sum error message, the error class messages (bit flags) and the relevant LEDs FK1, FK2 and FK3 are reset by power ON operation, for example. Refer to the previous chapters for further possibilities of resetting/acknowledging them.
- The error identification and the detailed information (word flag) are to be deleted by means of the user program or using the online function "Overwrite". They are also deleted when a cold start is performed or by a power-fail, if no backup battery is effective.

If an error has not been eliminated, the error message will appear again.

7.8 Additional diagnosis functions

7.8.1 CS31 status word EW 07,15 / %IW1007.15

The following information are continuously updated in the status word EW 07,15 / %IW1007.15:

- Bit 0: This bit is valid for the standalone PLC, for the Master PLC and for the slave PLC.
Bit 0 = 1 (TRUE), there is no class 2 error present.
Bit 0 = 0 (FALSE), there is a class 2 error.
- Bit 1: This bit is valid only for the slave PLC.
Bit 1 = 1 (TRUE), the slave PLC is added to the bus cycle of the master PLC.
Bit 1 = 0 (FALSE), the slave PLC is not added to the bus cycle of the master PLC.
- Bit 2 = 1 (TRUE), time and date are valid
- Bit 3 = 1 (TRUE), battery available
- Bit 4..7 are unused
- Bits 8...15: Maximum number of remote modules which have been existing in the CS31 bus cycle of the master PLC since the last power ON or the last cold start. This number may be larger than the number of remote modules which are currently existing in the CS31 bus cycle.

7.8.2 Capacity utilization of the basic unit and display of cycle time

For monitoring the capacity utilization of the basic unit, the following system flags are available:

- MW259,00 / %MW1059.0 current capacity utilization of the CPU in [%]

The overall capacity utilization of the CPU is displayed. I.e. it is included the system's basic utilization (task changes, internal error monitoring), the user tasks, the communication via ARCNET, serial interfaces, CS31 bus, PROFIBUS-DP etc.

- MW259,01 / %MW1059.1 Number of the task, which exceeded the cycle time

The number of the time-cyclic task is displayed, which cannot meet its configured cycle time. The numbering follows the entry in the task configuration of the 907 AC 1131.

- MW259,02 / %MW1059.2 Time in [ms] the task call came too late.
The time for the task is displayed and entered into MW259,01 / %MW1259.1.

- MW259,03 / %MW1059.3 Number of missed task calls.
The sum counter for all PLC user tasks is displayed.

The reaction to the cycle time exceeding is configured with the system constants:
- KW80,00 / %MW3080.0 - Monitoring the task with cycle time >0 and
- KW80,01 / %MW3080.1 - Monitoring the task with cycle time =0 (run-around).

In order to display the cycle times of the different tasks, the **PLC browser** in the object file "Resources" of the 907 AC 1131 can be used. Entering the command "tsk<ENTER>" in online mode, the current, minimum, maximum and average cycle time per user task is displayed.

7.8.3 Monitoring the floating-point functions

As of PLC runtime system **V4.16** there is the new system constant KW81,08 / %MW3081.8 to set the reaction to floating-point exceptions (see 2.5.9 Operands of the basic units / system constants).

The following applies:

KW81_08 AT %MW3081.8 : INT := 0; (* default *)

- The FK2 error $182_{Dec} = B6_{Hex}$ is output with a floating-point exception (e.g. with SQRT(-1)). The PLC is set to STOP. Using the call-up hierarchy of 907 AC 1131, it can be found out, where the error has occurred.

KW81_08 AT %MW3081.8 : INT := 1;

No error is output with a floating-point exception. Using the function block FPUXINFO (in SystemInfo_S90_V42.LIB), it can be found out, whether or not a floating-point exception occurred during the processor's calculation. With this information, it can be continued with default values or the machine can be shut down intentionally.

Programming example:

Under "Global Variables" / "System constants", the initialization value of KW81_08 is set to 1:
KW81_08 AT %MW3081.8 : INT := 1; (* Do not trigger off an FK2 error *)

```
PROGRAM PLC_PRG
VAR
FPUXINFO1 : FPUXINFO;
rV1 : REAL := -1.0;
rV2 : REAL;
bError : BOOL;
bWarning : BOOL;
END_VAR
```

```
bWarning := bError := FALSE;
(* Floating-point calculation *)
rV2 := SQRT(rV1);
(* Check, whether an exception has occurred *)
FPUXINFO1();
IF FPUXINFO1.ERR=TRUE THEN
(* here: the evaluation of the exception *)
(* e.g. shut down the machinery, continue calculation with standard values or corrected values *)
```

```
rV1 := 1.0;
bWarning := TRUE;
(* same calculation with corrected values *)
rV2 := SQRT(rV1);
(* check again.. *)
FPUXINFO1();
IF FPUXINFO1.ERR = TRUE THEN
    bError := TRUE;
END_IF
END_IF
(* here: e.g. evaluation bWarning, bError.. *)
```

7.9 Meaning of the contents of the error word flags

Explanation for the following table:

- Address = Memory address where the error was detected.
- Group number = Module address of the remote module.
- Channel number = Number of the faulty channel.
- Module type Meaning
 - 000 Digital input
 - 001 Analog input
 - 002 Digital output
 - 003 Analog output
 - 004 Digital input/output
 - 005 Analog input/output
 - 255 Bus master- or slave basic unit, where the error has occurred and where it is stored.

7.9.1 FK1-Fatal errors

Error class	Error description	Error identifier in MW 254,00/ %MW1254.0		Detailed info 1 in MW 254,01/ %MW1254.1	Detailed info 2 in MW 254,02/ %MW1254.2	Detailed info 3 in MW 254,03/ %MW1254.3	Further detailed infos in MW 254,04/ %MW1254.4 : MW 254,07/ %MW1254.7
		Dec	Hex				
FK1	Checksum error of the system EPROM	1 _D	1 _H	-	-	-	-
Fatal error	Operating system of the basic unit is defective or a defected RAM is detected when a cold start is performed (complete RAM test)	2 _D	2 _H	Address	-	-	-

7.9.2 FK2-Serious errors

Error class	Error description	Error identifier in MW 254,08/ %MW1254.8 Dec Hex	Detailed info 1 in MW 254,09/ %MW1254.9	Detailed info 2 in MW 254,10/ %MW1254.10	Detailed info 3 in MW 254,11/ %MW1254.11	Further detailed infos in MW 254,12/ %MW1254.12 : MW 254,15/ %MW1254.15	
FK2 Serious error	Illegal master/slave identifier	129 _D 81 _H	-	-	-	-	
	A serious error has occurred when the CS31 bus interfacing was initialized. The CS31 bus processor does not give any response to the PLC side within the specified time.	130 _D 82 _H	-	-	-	-	
	The PLC is overloaded, the cycle time is too short.	131 _D 83 _H	-	-	-	-	
	Checksum error in the Flash EPROM	133 _D 85 _H	-	-	-	-	
	Hardware watchdog	134 _D 86 _H	-	-	-	-	
	The CS31 bus processor reports an error via EW 07,15 / %IW1007.15 bit 0. This bit is checked prior to each start of the PLC program.	136 _D 88 _H	-	-	-	-	
	An illegal value has been configured for specifying the size of the I/O area between the master PLC and the slave PLC (KW 00,10 / %MW3000.10 or KW 00,11 / %MW3000.11).	137 _D 89 _H	-	-	-	-	
	A serious error has occurred when the I/O processor was initialized. The I/O processor does not give any response to the PLC side within the specified time.	139 _D 8B _H	-	-	-	-	
	The PLC is overloaded, the flashing process is not started.	180 _D B4 _H	-	-	-	-	
	The program is too large, flashing process cannot be performed.	181 _D B5 _H	-	-	-	-	
	Exception in the runtime system	182 _D B6 _H	-	-	-	-	
	Error of internal coupler (except ARCNET)	200 _D C8 _H	For detailed information see "System technology of the internal couplers".				
	An unknown operator or block is detected in the user program during the runtime.	258 _D 102 _H	7 characters representing the block name				
	The CS31 bus processor does not work correctly. The life identifier does not change.	259 _D 103 _H	-	-	-	-	
	The I/O processor does not work correctly. The life identifier does not change.	260 _D 104 _H	-	-	-	-	

7.9.3 FK3-Light errors

Error class	Error description	Error identifier in MW 255,00/ %MW1255.0		Detailed info 1 in MW 255,01/ %MW1255.1	Detailed info 2 in MW 255,02/ %MW1255.2	Detailed info 3 in MW 255,03/ %MW1255.3	Further detailed infos in MW 255,04/ %MW1255.4 : MW 255,07/ %MW1255.7
		Dec	Hex				
FK3 Light error	Remote module disconnected.	15 _D	F _H	Module type	Group number	-	-
	CS31 bus error (no remote module on the system bus). Note: If there are only analog modules connected to the CS31 system bus, this error message may occur when the supply voltage is switched on although the analog modules have been correctly added to the CS31 bus cycle after a certain time. Reason: The analog modules only appear on the CS31 system bus after the quite long initialization phase is completed. During the initialization time the master PLC cannot recognize them.	16 _D	10 _H	-	-	-	-
	The Flash EPROM cannot be programmed.	128 _D	80 _H	Address of defective memory cell	-	-	-
	Flash EPROM cannot be deleted	129 _D	81 _H	Address of memory cell which is undeletable	-	-	-
	The PLC operation mode configured in the system constant KW 00,00 / %MW3000.0 has not yet been activated. Please activate it (see also system constant KW 00,00 / %MW3000.0).	130 _D	82 _H	Value of KW 00,00/ %MW3000.0 activated last	Value of KW 00,10/ %MW3000.10 activated last	Value of KW 00,11/ %MW3000.11 activated last	-
	Inserted SMC user program card has a wrong version identifier for the operating system.	150 _D	96 _H	-	-	-	-
	Inserted SMC firmware card has a wrong identifier of the module type.	151 _D	97 _H	-	-	-	-

Error class	Error description	Error identifier in MW 255,00/ %MW1255.0 Dec Hex	Detailed info 1 in MW 255,01/ %MW1255.1	Detailed info 2 in MW 255,02/ %MW1255.2	Detailed info 3 in MW 255,03/ %MW1255.3	Further detailed infos in MW 255,04/ %MW1255.4 : MW 255,07/ %MW1255.7		
FK3 Light errors	Error with Flash task	180 _D B4 _H	-	-	-	-		
	Not enough memory Alloc()	181 _D B5 _H	-	-	-	-		
	Error during write/read of the SMC card		182 _D B6 _H	Error code:	-	-	-	
				001 _D 001 _H	Block number and number of blocks greater than max. allowed			
				002 _D 002 _H	Sector number greater than max. allowed			
				004 _D 004 _H	Block cannot be programmed, or cheksum error			
				008 _D 008 _H	Block already contains data			
				016 _D 010 _H	Sector cannot be erased			
				032 _D 020 _H	Card not initialized			
				064 _D 040 _H	Card not inserted or wrong ID code			
				128 _D 080 _H	Block is empty			
Error of internal couplers (except ARCNET)	200 _D C8 _H	Detailed information in ,System technology of the internal couplers'						
Routing error	300 _D 12C _H	004 _D 004 _H	Timeout counter					

7.9.4 FK4-Warning

Error class	Error description	Error identifier in MW 255,08/ %MW1255.8		Detailed info 1 in MW 255,09 %MW1255.9	Detailed info 2 in MW 255,10/ %MW1255.10	Detailed info 3 in MW 255,11 %MW1255.11	Further detailed infos in MW 255,12/ %MW1255.12 : MW 255,15/ %MW1255.15
		Dec	Hex				
FK4 Warning	Internal error of a remote module	1 _D	1 _H	Module type	Group number	Channel number	-
	Cut wire (open circuit)	2 _D	2 _H	Module type	Group number	Channel number	-
	Incorrect level on an analog output	3 _D	3 _H	Module type	Group number	Channel number	-
	Overload	4 _D	4 _H	Module type	Group number	Channel number	-
	Overload + cut wire	6 _D	6 _H	Module type	Group number	Channel number	-
	Short-circuit	8 _D	8 _H	Module type	Group number	Channel number	-
	Cut wire (at analog modules)	9 _D	9 _H	Module type	Group number	Channel number	-
	Short-circuit + cut wire "out of range" at analog modules	10 _D	A _H	Module type	Group number	Channel number	-
	Overload + short-circuit	12 _D	C _H	Module type	Group number	Channel number	-
	Short-circuit + overload + cut wire	14 _D	E _H	Module type	Group number	Channel number	-
	Internal error (non-maskable internal interrupt occurred)	136 _D	88 _H	-	-	-	-
	Internal error (an inhibited interrupt occurred)	137 _D	89 _H	-	-	-	-
	The module uses default adjustment values for the direct analog inputs and outputs instead of the factory settings.	140 _D	8C _H	-	-	-	-
Error of internal couplers (except ARCNET)	200 _D	C8 _H	Detailed information in 'System technology of the internal couplers'				

7.10 Reaction of the bus master basic unit and the remote modules in case of errors

No.	Error	Display/reaction of the bus master basic unit	Display/reaction of the input/output remote modules	Display/reaction of the slave basic units
1	Bus master basic unit has failed, e.g. due to a power failure.	No display, all outputs are set to FALSE (0).	LED (3) lights up. All outputs change to FALSE (0).	07 KR 91 / 07 KT 9x: - LED BA lights up LED RE flashes - Bit 1 = 0 (FALSE) in the status word EW 07,15/%IW1007.15 07 KR 31 / 07 KT 31: - Error LED flashes - Bit 1 = 0 (FALSE) in the status word EW 07,15/%IW1007.15
2	The bus master function of the basic unit (Serial unit) has failed, e.g. the bus processor is defective.	Displays: FK2 = Serious Error SE = Serial Unit Error Flags: M255,10/%MX255.10 = 1 M255,12/%MX255.12 = 1 Further flags, see 7.6		
3a or 3b	The CS31 system bus including at least 2 remote modules is disconnected (all remote modules are disconnect.) the CS31 system bus is short-circuited.	Displays: FK3 = Light Error BE = Bus Error RE = Remote Unit Error SE = Serial Unit Error Flags: M255,10/%MX255.10 = 1 M255,13/%MX255.13 = 1 Further flags, see 7.6		
4a	The CS31 system bus is disconnected (some of the remote modules are disconnected).	Displays: FK3 = Light Error BE = Bus Error RE = Remote Unit Error Flags: M255,10/%MX255.10 = 1 M255,13/%MX255.13 = 1 Further flags, see 7.6	Remote modules without connection to the bus master basic unit: reaction see No. 1	Slave basic units without connection to the bus master basic unit: reaction see No. 1
4b			Remote modules with connection to the bus master basic unit: no display/no reaction	Slave basic units with connection to the bus master basic unit: no display/no reaction
5a	The master basic unit does not find any remote modules on the CS31 bus after power ON or after cold start or CS31 system bus with at least 3 remote modules: 2 remote modules are disconnected.	Displays: BE = Bus Error Flags: M255,10/%MX255.10 = 1 M255,13/%MX255.13 = 1 For further flags, see 7.6	Remote modules with connection to the bus master basic unit: no display/no reaction	Slave basic units with connection to the bus master basic unit: no display/no reaction
5b	No connection to the CS31 system bus.		Remote modules without connection to the bus master basic unit: reaction see No. 1	Slave basic units without connection to the bus master basic unit: reaction see No. 1
5c	Defective remote modules.		Not clear	Error class FK1 / FK2, all outputs change to FALSE (0).
5d	Power supply failure.		All outputs change to 0 0	All outputs change to FALSE (0)
6a	An error has occurred at the inputs or outputs of a remote module, e.g. a short-circuit.	RE = Remote Unit Error Flags: M255,10/%MX255.10 = 1 M255,14/%MX255.14 = 1 (FK4)	Involved remote module: LED (3) lights up The LEDs (1) provide detailed information when using the test button(4).	Involved 07 KT 9x: LED Ovl. = Short-circuit 07 KT 31: Error LED ON Flags (07 KT 9x, KT 31): M255,10/%MX255.10=TRUE(1) M255,14/%MX255.14=TRUE(1) For further flags see 7.6
6b			Not involved remote mod.: no display/no reaction	Not involved slave basic units: no display/no reaction

Reaction of the bus master basic unit and the remote modules in case of errors (continued)

No.	Error	Display/reaction of the bus master basic unit	Display/reaction of the input/output remote modules	Display/reaction of the slave basic units
7a	Two remote modules of the same type with inputs are set to the same address.	<p>This error is detected only when the signal states of two modules become different. In this case, the telegram is faulty and the modules are considered as disconnected.</p> <p>Display: RE = Remote Unit Error</p> <p>Flags: M255,10/%MX255.10 = 1 M255,13/%MX255.13 = 1 For further flags see 7.6</p>	<p>Involved modules: reaction see No. 1</p> <hr/> <p>Other modules: no display/no reaction</p>	Reaction see No. 1
7b	Two remote modules of the same type are set to the same address.	No reaction, unless there is a large distance between the remote modules.	Error-free operation of the two modules, unless there is a large distance between the modules.	Not applicable because inputs and outputs are always present.
7c	Two remote modules of different types, but with overlapping ranges are set to the same address, e.g. ICSI 16 D1 and ICSK 20 F1.	The error is already detected during initialization. The remote modules are not added into the bus cycle.	Involved modules: reaction see No. 1	Involved modules: reaction see No. 1
			Other modules: no display/no reaction	Other modules: no display/no reaction
7d	Address 62 or 63 has been set for a binary remote module.	Is not detected.	<p>- Signal output in parallel to the bus master</p> <p>- Input signals are ignored.</p>	-
7e	An address > 5 has been set for an analog remote module.	Is not detected.	Reaction see No. 1	-

8.1 Operating modes of the serial interfaces

Interface standard: EIA RS-232

The interface operating mode is set depending on the particular application.

Programming and test

or

Man-Machine Communication MMC

Active mode: The active mode is used for programming and testing the basic unit, i.e. it provides access to all programming and test functions of the basic unit.

Passive mode: The passive mode serves for the communication between the user program and a module which is connected to the serial interface (e.g. MODBUS protocol).

8.2 Basic features of the serial interfaces

The serial interfaces of the 07 KT 97 can be used in two different modes: active mode and passive mode. Active mode always means the programming and online access with 907 AC 1131. Passive modes are such interface modes which are set up by means of a function block.

The interfaces can be set to the modes independently of each other. The passive modes are installed by function blocks (e.g. MODINIT, COMINIT) from the user project. The selection of the interface number is performed automatically. It is explained in the following sections.



Caution:

Under no circumstances, both serial interfaces can be set to the active mode **at the same time**.

- Both interfaces are able to support the active mode, however, only one at a time.
- Both interfaces are able to support the passive mode. It is possible to use them in passive mode at the same time. The combination of the operating modes MODBUS slave, MODBUS master and 'free mode' COM is no subject to any restrictions.
- In order to operate an interface in the passive mode, a connection between the pins 6 and 8 is necessary at the interface cable on the PLC side.
- Per serial interface, only one passive mode initialization function block may be used (e.g. 1 x MODINIT or 1 x COMINIT).
- In the MODBUS Slave operating mode, the driver functionality is completely realized by the MODINIT function block.
- In the MODBUS Master operating mode, transmitting and receiving of messages is handled by the additional function block MODMAST. It is recommended to use only one MODMAST block for each interface within one project. If necessary, the input values can be changed during the course of the program according to the requirements. If several MODMAST blocks are used, it is absolutely necessary to lock the enabling inputs of the blocks against one another.

- In the free operating mode COM, only one COMREC receiving block may be used.
- In the free operating mode COM, several COMSND blocks may be used for each interface.

8.3 Behaviour of the serial interfaces

The serial interfaces have several automatic functions concerning setting and changing the operating mode. The resulting functionality is explained in the following chapters.

8.3.1 Starting-up / Booting the PLC

While starting up the PLC, at first the serial interfaces are checked **for connections between the pins 6 and 8**. In case of connected cables (e.g. MODBUS via COM1) the valid programming interface is detected in this way.

The following table shows the automatically set programming interface detected while starting up the PLC depending on the connected interface cables.

COM1	COM2	Programming access via
pins 6/8: no connection	pins 6/8: no connection	COM1
pins 6/8: no connection	pins 6/8: connection	COM1
pins 6/8: connection	pins 6/8: no connection	COM2
pins 6/8: connection	pins 6/8: connection	COM1

If cables with connections between pins 6 and 8 are installed at both serial interfaces, COM1 will automatically set as the programming interface at first while starting up the PLC. This makes sure that under all conditions a programming access is available. In order to establish a connection between 907 AC 1131 and the PLC, the cable containing the connection between the pins 6 and 8 has to be replaced by a suitable programming cable.

8.3.2 Configuration by using function blocks in the user program

If an interface initialization block (e.g. MODINIT, COMINIT) has successfully re-configured the currently valid serial programming interface, it automatically checks, if the other of the two interfaces is occupied by another initialization block. If this interface is free, it will be configured as the programming interface.

Example:

While starting up the PLC, no connection was detected between the pins 6 and 8 at the COM1 interface. As a consequence, COM1 will be configured as the programming interface.

In the user program, a COMINIT block is successfully used with COM1 (all input parameters have valid values, there is a connection between the pins 6 and 8 of COM1). As a result, COM1 will be configured in the free mode and COM2 as a programming interface.

Later on, the user program applies a MODINIT block successfully on COM2 (all input parameters have valid values, there is a connection between the pins 6 and 8 of COM2). From now on, COM1 will continue to be in the free mode and COM2 will run as a MODBUS interface. Now, both interfaces are occupied and, as a consequence, there is no longer a programming interface.

8.3.3 PLC RUN->STOP

If none of the two serial interfaces is available as programming interface, then, in case of STOP condition, the programming interface is automatically re-configured to that serial interface which was set with the start-up of the PLC, regardless of a connection between the pins 6 and 8 (see example above, last paragraph). In this case, the parameters (e.g. baud rate etc.) required for the access to 907 AC 1131 are also restored.

8.3.4 Removing an existing connection between the pins 6 and 8 of the serial interface during a running user program

If an interface was successfully configured in the user program in passive mode with MODINIT or COMINIT and, if this initialization block is still enabled (EN = TRUE, normal condition), the following applies:

The blocks automatically detect if the cable is unplugged (connection between pins 6 and 8 is missing) and signalize a corresponding error. If there is no serial interface configured as programming interface in this moment (both interfaces are used in passive mode by MODINIT and/or COMINIT) the block makes the interface the programming interface temporarily. The required parameters (baud rate etc.) are also set then.

This function is especially useful for the test of programs at PLCs without ARCNET connection and with both serial interfaces used in passive mode.

If the program was not modified (only monitored, no download, no online change), the block continues the previously configured normal operation after re-plugging the cable containing the connection between the pins 6 and 8. The pre-set parameters are still used. Otherwise the program must be re-started.



Note:

This function is supported by MODBUS blocks with run-time system versions 4.04 and later. In older run-time systems, the MODBUS blocks do not support the functions described above.

8.3.5 Automatic 907 AC 1131 login detection

As an option, the automatic 907 AC 1131 login detection can be activated for a serial interface in passive mode. Normally, the automatic login detection is deactivated. It can be activated by means of the COMAUTOLOGIN function from the COM_S90_V41.LIB library.

It is recommended, to activate the automatic login detection only with those projects which need it explicitly. Communication via the serial interface runs slower with the automatic login detection activated.

If the login detection is activated, the 907 AC 1131 login procedure via a serial interface in passive mode is automatically detected, regardless of a connection between the pins 6 and 8. This is very useful, if, for instance, at first a modem connected to a serial interface has to be initialized and set into operation by means of the blocks COMINIT, COMSND and COMREC, in order to establish a remote maintenance access via 907 AC 1131 during running operation.



Note:

If it was logged in with 907 AC 1131, this is output at the COMINIT block with ERNO=65 or at the MODINIT block with ERNO=50.

The login procedure of 907 AC 1131 V4.0 is detected via the following character string:

AA _{HEX}	AA _{HEX}	01 _{HEX}	00 _{HEX}	01 _{HEX}	00 _{HEX}	58 _{HEX}	01 _{HEX}	01 _{HEX}	-	-
BLOCK IDENT (Low-Byte)	BLOCK IDENT (High-Byte)	BLOCK SIZE (Low-Byte)	BLOCK SIZE (High-Byte)	BLOCK NUMBER (Low-Byte)	BLOCK NUMBER (High-Byte)	CHECK SUM	LAST BLOCK	SERVICE ID (LOGIN)	Info 1	Info 2

The login procedure of 907 AC 1131 V4.1 (and later) is detected via the following character string:

AA _{HEX}	AA _{HEX}	01 _{HEX}	00 _{HEX}	01 _{HEX}	00 _{HEX}	58 _{HEX} 59 _{HEX} 59 _{HEX} 5A _{HEX}	01 _{HEX}	01 _{HEX}	00 _{HEX} 01 _{HEX} 00 _{HEX} 01 _{HEX}	00 _{HEX} 00 _{HEX} 01 _{HEX} 01 _{HEX}
BLOCK IDENT (Low-Byte)	BLOCK IDENT (High-Byte)	BLOCK SIZE (Low-Byte)	BLOCK SIZE (High-Byte)	BLOCK NUMBER (Low-Byte)	BLOCK NUMBER (High-Byte)	CHECK SUM	LAST BLOCK	SERVICE ID (LOGIN)	Info 1	Info 2



Note:

Make sure that this character string does not appear within a normal communication message if the automatic login detection is activated. Otherwise, the character string will be interpreted as the 907 AC 1131 login character string. This can cause an unintended malfunction of the application.



Note:

The automatic login detection at an interface in passive mode is only possible, if the corresponding initialization block (COMINIT or MODINIT) is active for this interface (EN = TRUE).

**Note:**

The login detection for 907 AC 1131 V4.0 is supported by the COMINIT and. MODINIT blocks as of runtime system version V4.05.

The login detection for 907 AC 1131 for 907 AC 1131 V4.1 (and later) is supported by the COMINIT and. MODINIT blocks as of runtime system version V4.15.

The possibility of the login detection depends also on the type of connection between the PLC and 907 AC 1131.

If the connection is established directly via RS-232, a login message can only be detected if during initialization of the interface the same parameters were set as used by 907 AC 1131 (19200 baud, 1 stop bit, no parity bit, character length 8 bits).

If the connection is established via an RS232/RS485 interface converter, a login message can also only be detected if the initialization parameters for the passive mode are equal to the parameters used by 907 AC 1131. Since in such an application normally several participants are connected to the RS-485 transmission line, also the following has to be taken into account: The 907 AC 1131 login message does not include an address of the participant. For that reason, all participants on the RS-485 transmission line will detect the login message, provided that they are programmable with 907 AC 1131 and that their interface can interpret the login message (interface in active mode or passive mode with activated COMAUTOLOGIN). During the following acknowledgment of the login message by these participants message collisions can occur. As a consequence, the communication may be aborted.

If the connection between 907 AC 1131 and the PLC is established via modem, the communication is not influenced by the interface parameters set by COMINIT and MODINIT. The parameters, required by the modem, are to be set. After initialization, the modem converts the received data according to the settings. The assignment of the login message to a certain PLC is also guaranteed, because the connection only can be established by the modem-allocated telephone number or by MSN.

Logging-in with 907 AC 1131 causes a re-initialization of the interface at first. All blocks which access this interface are locked during the length of the online session, i.e. they do not perform any operation. The outputs of the blocks have the following values during this time:

RDY = FALSE
ERNO = COM_AC1131_REMOTE_ACCESS

After logging-out with 907 AC 1131 the blocks are activated again.

The login monitoring for an interface only takes place, if 907 AC 1131 has not been logged-in via an other interface before (e.g. ARCNET, other COM). In addition, only such interfaces are monitored which are configured in a passive mode.

8.4 Interface parameters

Active mode:

The interface parameter settings are fixed and cannot be changed.

Data bits:	8
Stop bits:	1
Parity bit:	none
Transfer rate:	19200 baud
Synchronization:	RTS/CTS

Passive mode:

The interface parameter settings are performed by the respective function block.

Data bits:	5, 6, 7 or 8
Stop bits:	1, 1,5 (for 5 data bits) , 2 (for 6, 7 or 8 data bits)
Parity bit:	none
Transfer rate:	300, 1200, 2400, 4800, 9600, 14400 oder 19200 Baud
Synchronization:	none direction control signal RTS optionally with carrier leading time and carrier lagging time see initialization blocks (e.g. COMINIT in COM_S90_V4x.LIB ab V4.1)

9.1 Programming system 907 AC 1131

The programming system 907 AC 1131 is used to program all versions of the controllers 07 KT 95, 07 KT 96, 07 KT 97 and 07 KT 98. The controllers 07 KR 91, 07 KT 92, 07 KT 93, 07 KT 94, 07 Kx 3x, 07 Kx 4x and 07 Kx 5x **cannot** be programmed using the 907 AC 1131.

The software can be executed on IBM/AT compatible Personal Computers having the operating system **Windows NT (version 4.0 or later, service pack 4 or higher), Windows 98 SE (Second Edition) und Windows 2000** installed. The drivers necessary for the serial communication and the communication via ARCNET are provided with the programming system.

The software package 907 AC 1131 is installed on the PC by a installation program which is extensively automated (see volume 6 / "Installation"). The ARCNET drivers have to be installed additionally.

Apart from the usual functionality, the 907 AC 1131 presents the following outstanding features:

- Supporting the PLC languages:
 - Function Block Diagram (FBD)
 - Ladder Diagram (LD)
 - Instruction List (IL)
 - Sequential Function Chart (SFC)
 - Structured Text (ST)
- Simple handling
- Rapidity and low use of resources
- Very short compilation times
- Compact project files (one program file per project)
- Complete and practical operation via keyboard, Particularly for the graphic editors
- Many powerful commissioning features:
 - Monitoring of variable values
 - Writing and forcing of variable values
 - Single cycle
 - Breakpoints, single-step operation and calling stack
- Sampling Trace: cycle-exact recording and graphical display of variable values
- Powerflow: display of passed lines, display of the accumulator contents in IL or of the lione status in FBD
- Integrated visualization for commissioning, training, service and offline tests
- Offline simulation (not for external libraries, such as the ABB libraries)
- Online modifications (changes can be performed during operation)
- Conversion between the PLC languages (IL, FBP, LD)
- Extensive support of the following IEC types:
 - BOOL, 8, 16 and 32-bit integer values, 32 and 64-bit reals, strings, time, date, time of date, arrays (also multi-dimensional), structures, alias types, any combinations of structures, arrays and the remaining data types

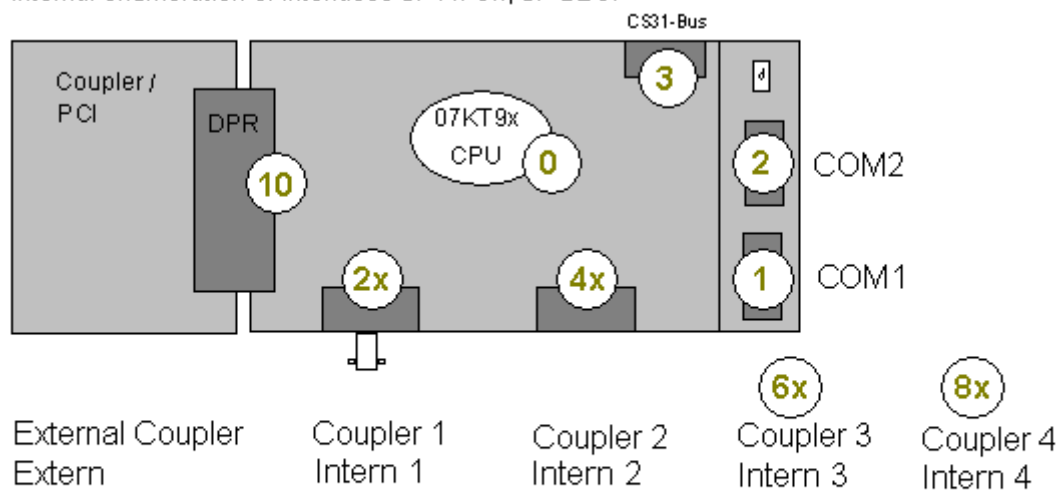
The special functions of 907 AC 1131 for the 40/50 control system series are described under volume 8.

9.2 Variants of programming the 07 KT 97 with 907 AC 1131

In order to communicate with the "external world", the PLCs 07 KT 95..98, 07 SL 97 have the following interfaces:

No.	Term	Interface	Programming access
0	CPU	CPU of its own	CPU for online operation
1	COM1	Serial interface COM1	yes
2	COM2	Serial interface COM2	yes, if COM1 in passive mode free with 07 SL 97 !
3	-	DPR to CS31 bus processor	no
10	Extern	PCI-DPR interface	yes
20..39	Intern 1	Coupler in slot 1 with channel 0..19	depends on TYPE (z. B. ARCNET – yes, PROFIBUS DP – no)
40..59	Intern 2	Coupler in slot 2 with channel 0..19	depends on TYPE
60..79	Intern 3	Coupler in slot 3 with channel 0..19	depends on TYPE
80..99	Intern 4	Coupler in slot 4 with channel 0..19	depends on TYPE

Internal enumeration of interfaces 07 KT 9x, 07 SL 97



Example: Coupler on internal slot 1 with 2 channels

Intern 1: Channel 0 --> 20

Channel 1 --> 21

Address: dependend on coupler, max. 5 bytes

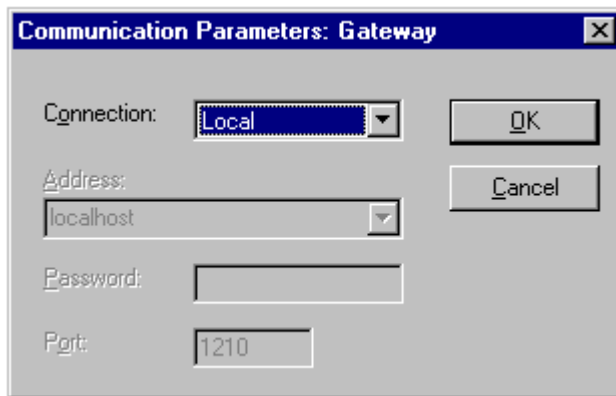
Communication drivers:

For programming the 07 KT 97, the following communication drivers are available:

ABB RS232	Driver for the serial interfaces
ABB RS232 Route	Driver for the serial interfaces with Routing (as of runtime system V4.16)
ABB Tcp/Ip Level 2	Ethernet driver with Routing (as of runtime system V5.0)
ABB Arcnet	Driver for programming via ARCNET
ABB Arcnet Route	Driver for programming via ARCNET with Routing (as of runtime system V4.15)
ABB Arcnet Route fast	Driver for programming via ARCNET with Routing and settable block size (as of runtime system V5.0)
ABB SL97	PCI driver for the 07 SL 97 Slot-PLC with Routing (as of runtime system V4.15x and operating system Win NT 4.0 SP 5)
ABB SL97 fast	PCI driver for the Slot PLC 07 SL 97 with Routing and settable block size (as of runtime system V5.0, operating system Win NT 4.0 SP 5)
Serial (Modem)	Modem driver for modem at a serial interface of the PC and the PLC
ABB Modem Route	Serial modem driver with Routing (as of runtime system V5.0)

Gateway setting:

Select the setting "Local" under "Online/Communication parameters/Gateway" in the 907 AC 1131 programming system (see Volume 10, Operating Manual 907 AC 1131 / Chapter 4 – The Individual Components / General Online Functions / Online Communication Parameters for the use of Gateway):



For the Ethernet driver "ABB Tcp/Ip Level 2" the following Gateway setting is valid:

Protocol: Tcp/Ip
Address: localhost
Port: 1210

Setting the communication parameters:

Setting the communication parameters as well as the address data is performed with the 907 AC 1131 programming software by choosing the appropriate driver and entering the data into the corresponding communication parameter mask in the menu "Online/Communication parameters".

In the following, the settings of the drivers mentioned above are described.

9.3 Programming via the serial interfaces

The operating modes of the serial interfaces COM1 and COM2 are described in chapter 8. The serial interface COM1 is used as the default programming interface (without MMC).

In volume 6 – Installation you can find information about setting the serial interfaces for the different PC operating systems.

The connection between PC and PLC is established with the 07 SK 90 system cable.

The settings of the interface parameters are fixed and cannot be changed.

Transmission speed: 19200 baud

Number of parity bits: none

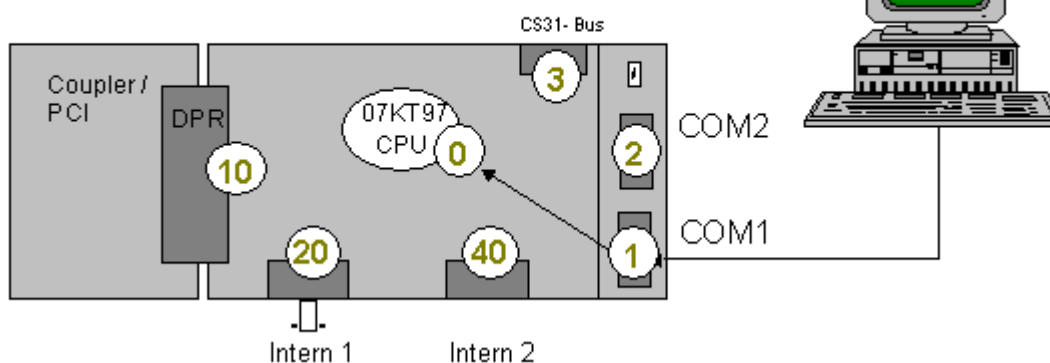
Number of data bits: 8

Number of stop bits: 1

Handshake RTS/CTS

PC at serial interface COM1 / online this CPU

Driver: ABB RS232 or ABB RS232 Route

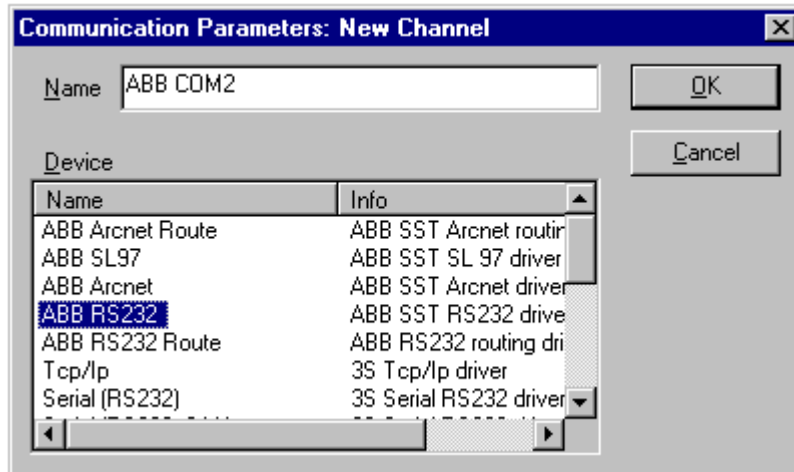


During the installation process of the 907 AC 1131 programming software, the Gateway channel "ABB RS232" is created for the serial PC interface COM1 with the "ABB RS232" driver.

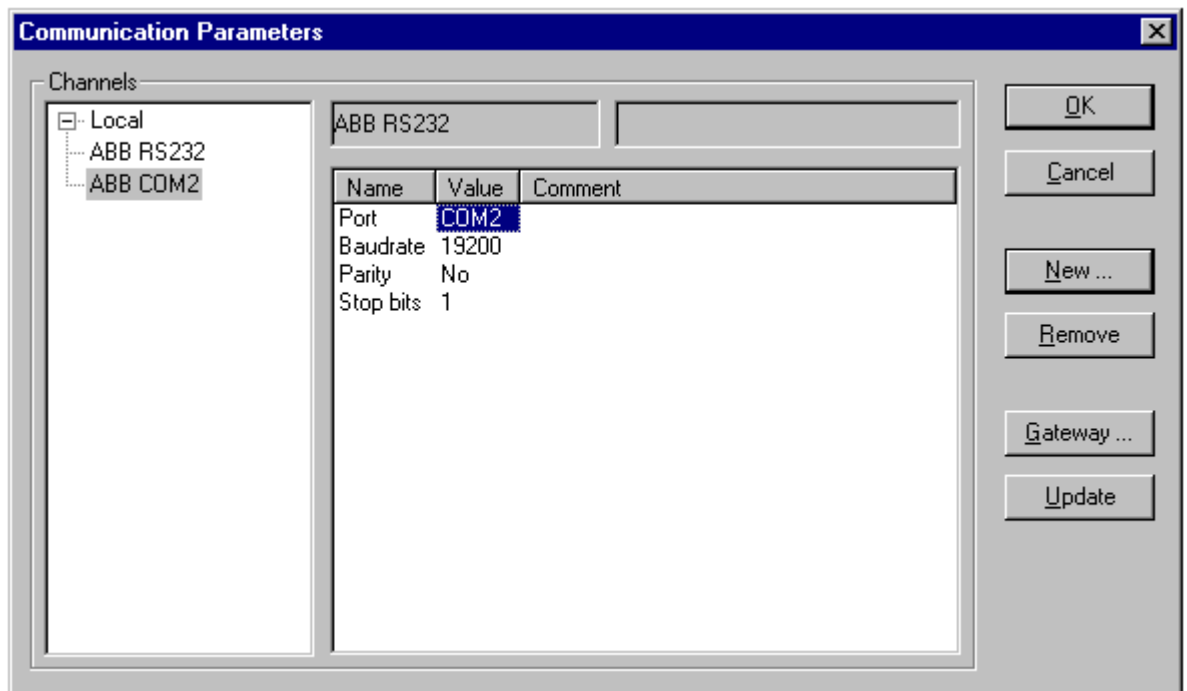
9.3.1 Serial driver "ABB RS232"

The serial driver „ABB RS232“ is the standard driver for programming via the serial interface of the PLC. When using this driver, no access is possible to the PLC with 907 FB 1131 or OPC. With this driver active, only one Gateway channel can be opened.

In order to establish a new Gateway channel for the serial interface, select "Online/Communication parameters/New", enter a name for the new channel (e.g. ABB COM2) and then select the "ABB RS232" driver.



For the serial driver "ABB RS232", the following communication parameter mask is used:



Parameter	Possible values	Meaning
Port	COM1, COM2 (PC-specific)	serial PC interface
Baudrate	19200	always 19200 baud
Parity	No	always no parity
Stop bits	1	always one stop bit

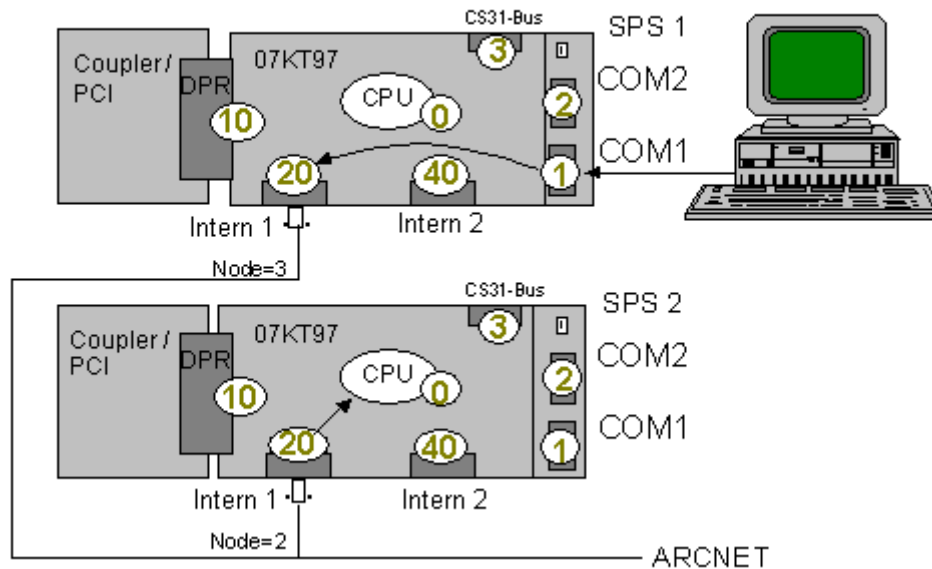
9.3.2 Serial driver "ABB RS232 Route"

The serial driver "ABB RS232" was introduced with the runtime system V4.0. As of **runtime system V4.16** there is the additional serial Routing driver "**ABB RS232 Route**". This driver offers:

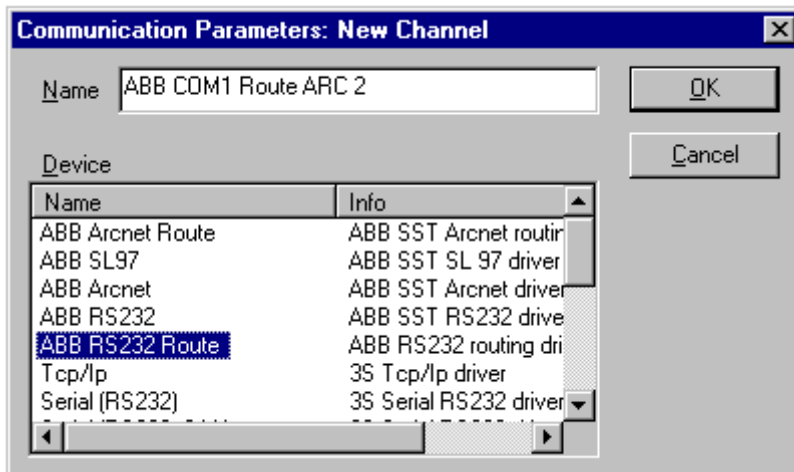
- Online operation of the PLC with 907 AC 1131
- Online operation of the PLC with 907 FB 1131
- OPC connection with OPC server as of V2.xx
- Online operation with PLCs, networked via ARCNET, using the serial interface
- Parallel operation of 907 AC 1131 and 907 FB 1131
- Parallel operation of 907 AC 1131 and OPC server

PC at serial interface COM1 and over Intern 1 at ARCNET / online ARCNET Node 2

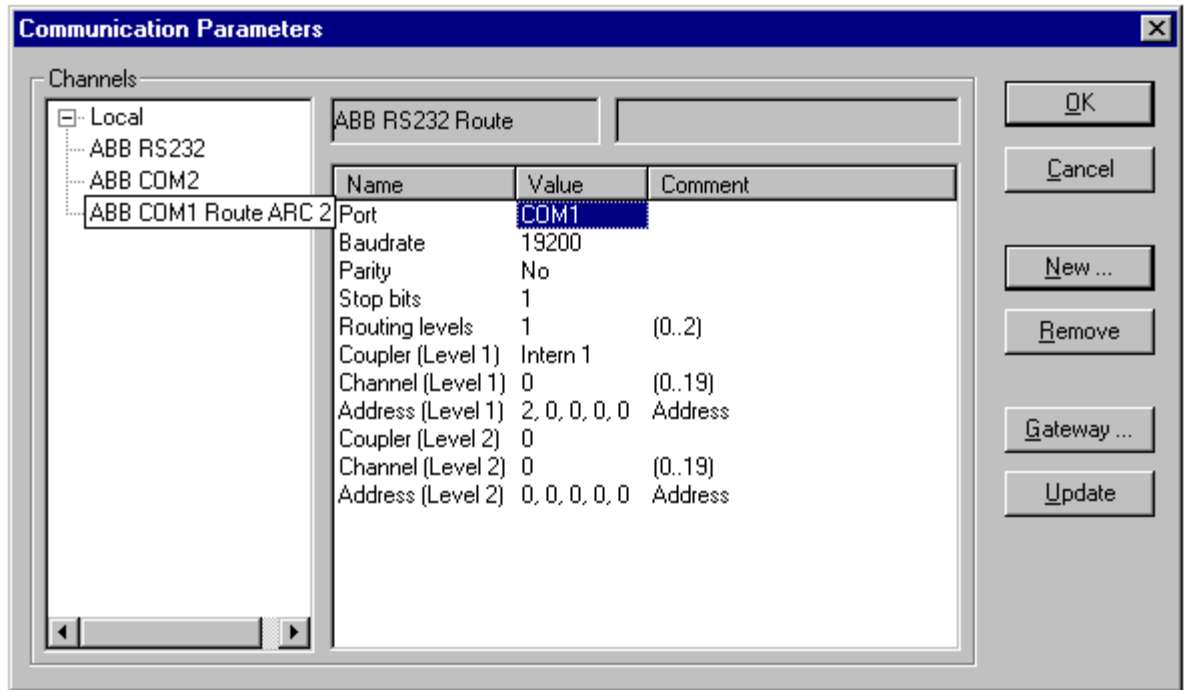
Driver: ABB RS232 Route



In order to establish a new Gateway channel for the serial Routing driver, select "Online/Communication parameters/New", enter a name for the new channel (e.g. ABB COM2 Route ARC 2) and then select the "ABB RS232 Route" driver.



For the serial Routing driver "ABB RS232 Route", the following communication parameter mask is used:



Parameter	Possible values	Meaning
Port	COM1, COM2 (PC-specific)	serial PC interface
Baudrate	19200	always 19200 baud
Parity	No	always no parity
Stop bits	1	always one stop bit
Routing levels	0..2	Routing levels (0 = none)
Coupler (Level 1)	0, Extern, Intern 1..Intern 4	coupler for level 1
Channel (Level 1)	0..19	channel to coupler level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 1
Coupler (Level 2)	0, Extern, Intern 1..Intern 4	coupler for level 2
Channel (Level 2)	0..19	channel to coupler level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 2

If it is required to directly access to the connected CPU with the serial Routing driver, all of the Routing parameters (as of Routing levels) must be entered with 0.

The ARCNET couplers of the PLCs 07 KT 97, 07 KT 98 and 07 SL 97 are always the coupler 1. Thus they are entered with "Intern 1".

The ARCNET coupler has only one communication channel. Therefore, "Channel" must always be entered with 0.

For the node address, one byte is necessary for the ARCNET coupler. The address is entered into the first byte of the address bytes.

For the Routing levels the following applies:

Routing levels = 0 no Routing (no entry of parameters for level1 and level 2)

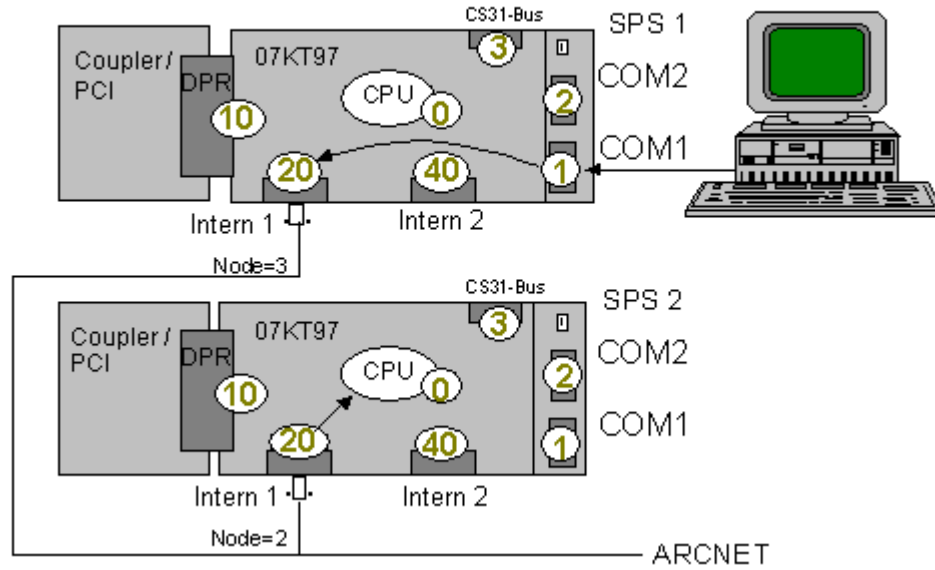
Routing levels = 1 one-level Routing (entry of parameters for level 1)

Routing levels = 2 two-level Routing (entry of parameters for level 1 and level 2)

Example:

PC at serial interface COM1 and over Intern 1 at ARCNET / online ARCNET Node 2

Driver: ABB RS232 Route



PLC 2 = ARCNET station 2 is to be programmed via PLC 1 = ARCNET station 1. The interface PC/COM2 is connected to the serial interface COM1 of the PLC 1:

Parameter	Value	Remarks
Port	COM2	PC interface COM2
Baudrate	19200	
Parity	No	
Stop bits	1	
Routing levels	1	one-level Routing
Coupler (Level 1)	Intern 1	coupler 1
Channel (Level 1)	0	channel 1
Address (Level 1)	2, 0, 0, 0, 0	node address of target PLC (node 2)
Coupler (Level 2)	0	no second level
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	

9.4 Programming via ARCNET

The programming via ARCNET is only possible on a PC with an ARCNET card included. The installation of the card(s) and drivers is described in Volume 6/ Installation.

When programming via ARCNET, the PC is one of the stations in the ARCNET net. As a default, the PC is given the ARCNET address 254.

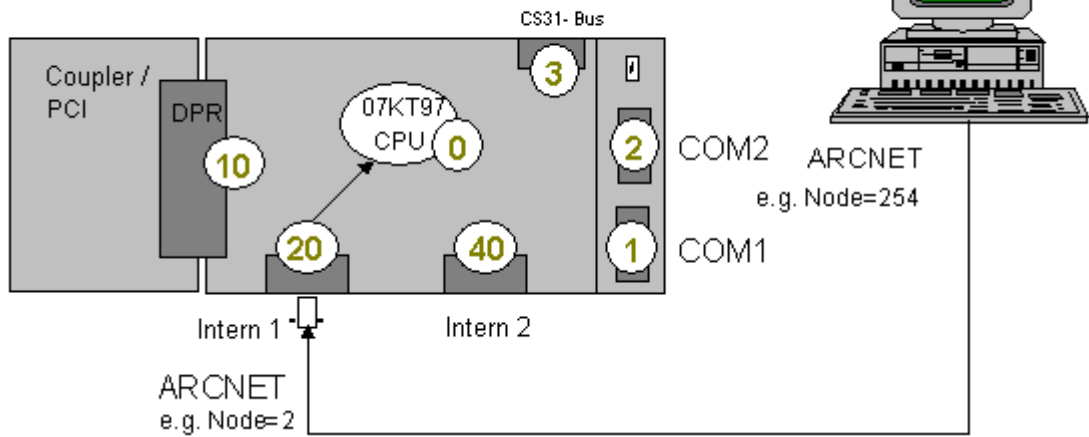


Caution:

Up to 907 AC 1131 V4.2, only one instance of 907 AC 1131 may be opened for communication with the PLC via ARCNET. As of V4.3 using the drivers „ABB Arcnet Route“ or „ABB Arcnet Route fast“, several instances of AC1131.EXE can communicate with the PLCs.

PC with ARCNET / online ARCNET Node 2

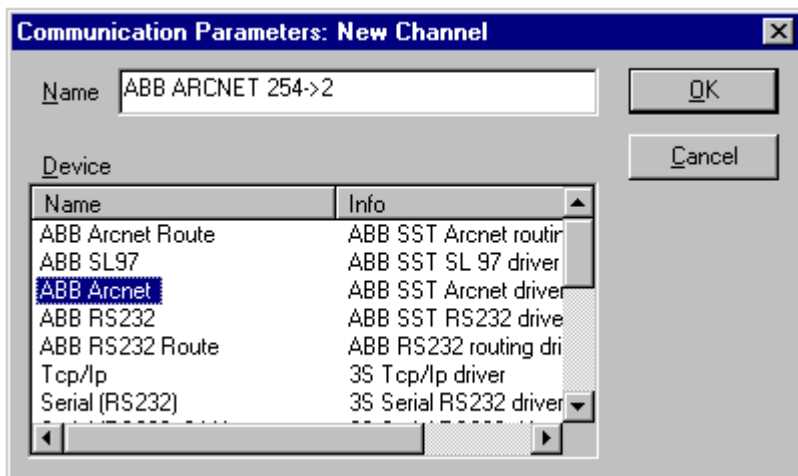
Driver: ABB Arcnet or ABB Arcnet Route



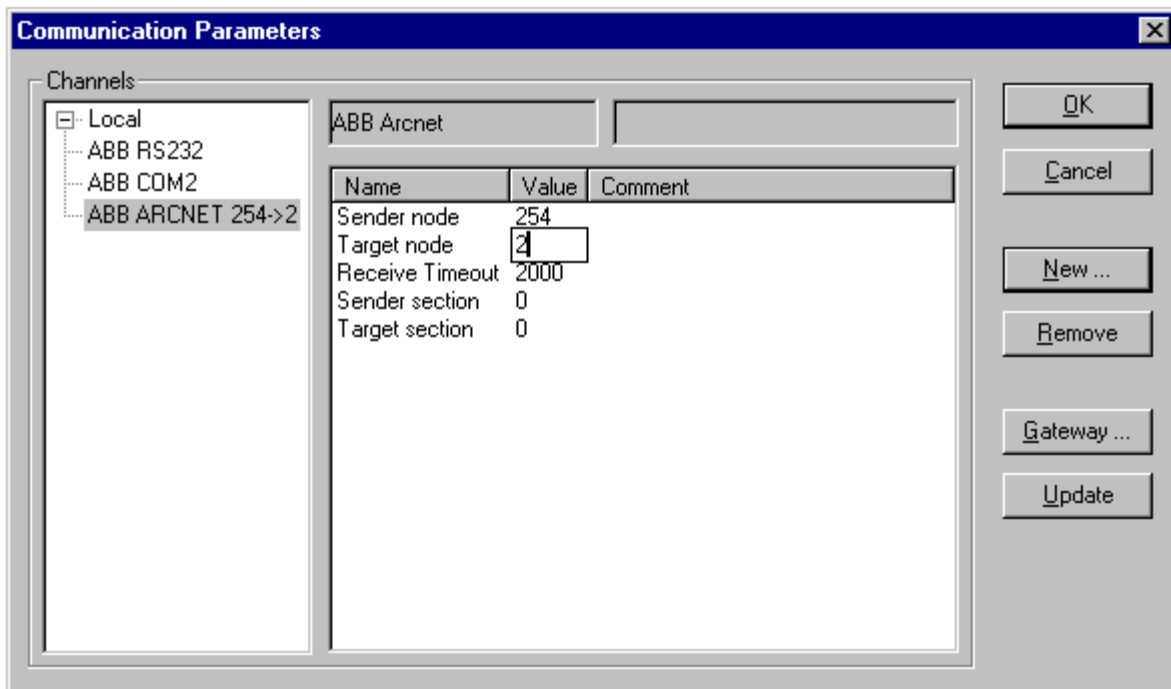
9.4.1 ARCNET driver "ABB Arcnet"

The ARCNET driver "ABB Arcnet" was introduced with the PLC runtime system V4.0.

In order to establish a new Gateway channel for the ARCNET driver, select "Online/Communication parameters/New", enter a name for the new channel (e.g. ABB ARCNET 254→2) and then select the "ABB Arcnet" driver.



For the ARCNET driver "ABB Arcnet", the following communication parameter mask is used:



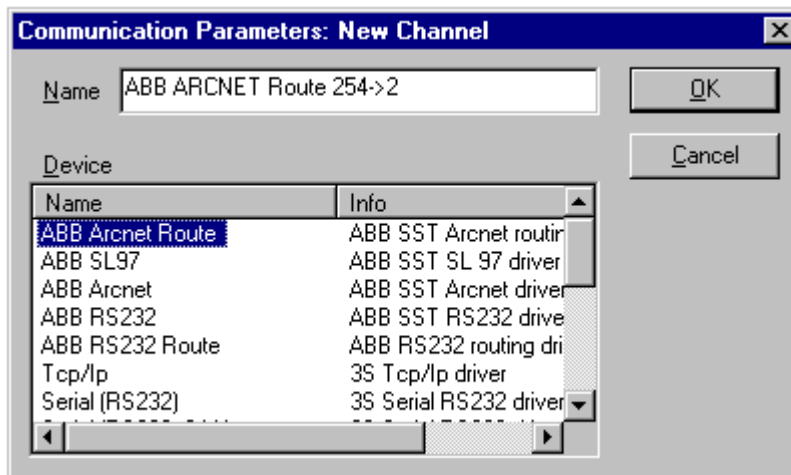
Parameter	Possible values	Meaning
Sender node	1..255 (default: 254)	ARCNET address of the PC
Target node	1..255	ARCNET address of the PLC
Receive Timeout	>= 2000	Timeout [ms] for response
Sender section	0	reserved
Target section	0	reserved

9.4.2 ARCNET driver "ABB Arcnet Route"

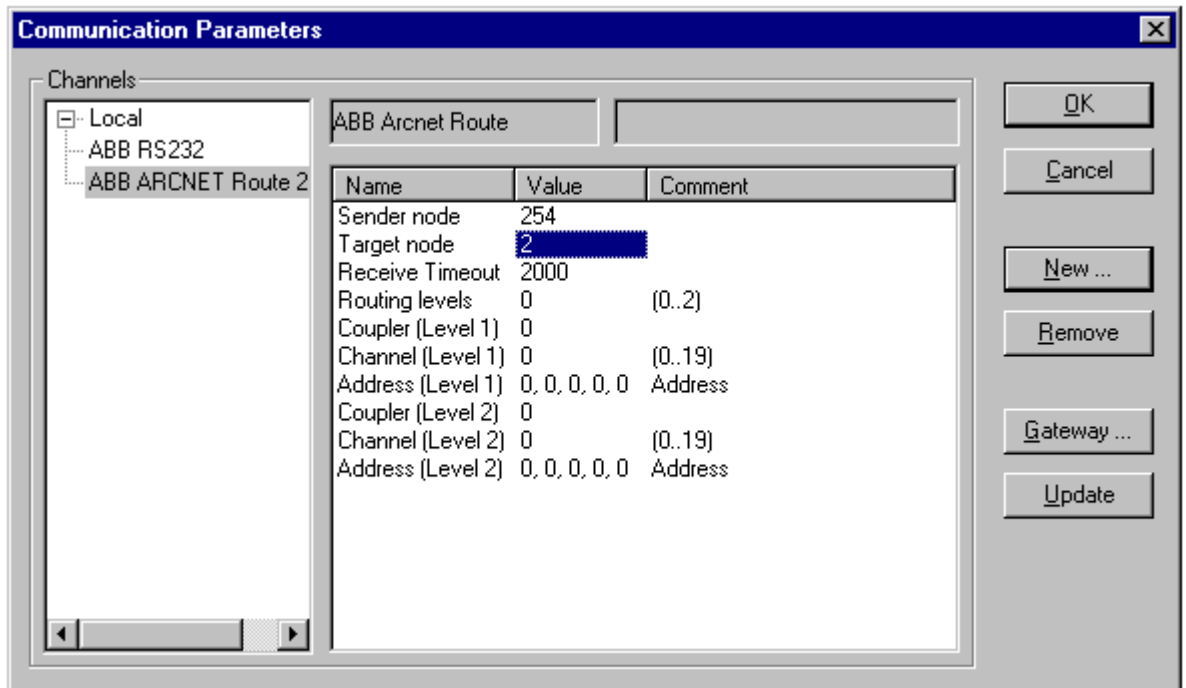
The ARCNET driver "ABB Arcnet" was introduced with the PLC runtime system V4.0. As of the **runtime system V4.15**, there is the ARCNET Routing driver "**ABB Arcnet Route**" additionally available. This driver offers:

- Online operation of the PLC with 907 AC 1131
- Online operation of the PLC with 907 FB 1131
- OPC connection with OPC server as of V2.xx
- Parallel operation of 907 AC 1131 and 907 FB 1131
- Parallel operation of 907 AC 1131 and OPC server
- Parallel operation instances of AC11341.EXE with several PLCs

In order to establish a new Gateway channel for the ARCNET Routing driver, select "Online/Communication parameters/New", enter a name for the new channel (e.g. ABB ARCNET Route 254→2) and then select the "ABB Arcnet Route" driver.



For the ARCNET Routing driver "ABB Arcnet Route", the following communication parameter mask is used:



Parameter	Possible values	Meaning
Sender node	1..255 (default: 254)	ARCNET address of the PC
Target node	1..255	ARCNET address of the PLC
Receive Timeout	>= 2000	Timeout [ms] for response
Routing levels	0..2	Routing levels (0 = none)
Coupler (Level 1)	0, Extern, Intern 1..Intern 4	Coupler for level 1
Channel (Level 1)	0..19	Channel to coupler level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 1
Coupler (Level 2)	0, Extern, Intern 1..Intern 4	Coupler for level 2
Channel (Level 2)	0..19	Channel to coupler level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 2

If it is required to directly access to the connected CPU with the ARCNET Routing driver, all of the Routing parameters (as of Routing levels) must be entered with 0.

The ARCNET couplers of the PLCs 07 KT 97, 07 KT 98 and 07 SL 97 are always the coupler 1. Thus they are entered with "Intern 1".

The ARCNET coupler has only one communication channel. Therefore, "Channel" must always be entered with 0.

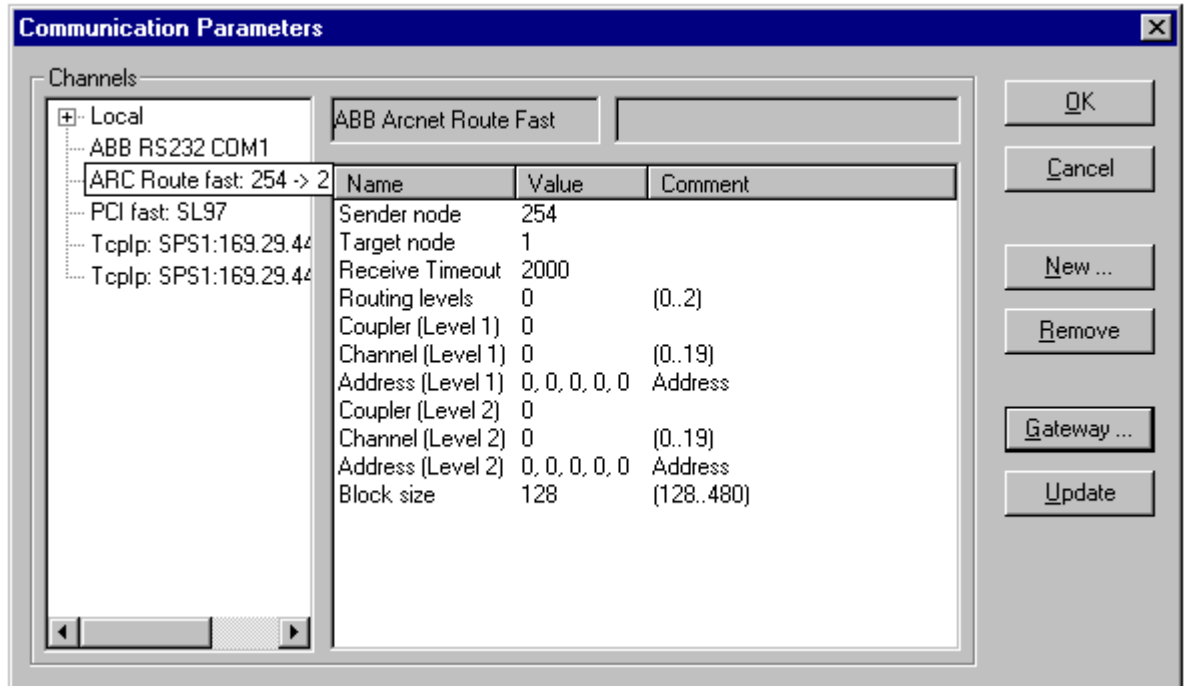
For the node address, one byte is necessary for the ARCNET coupler. The address is entered into the first byte of the address bytes.

9.4.3 ARCNET Driver "ABB Arcnet Route fast"

As of runtime system V5.0 of the PLC and 907 AC 1131 V5.0, the additional driver "ABB Arcnet Route fast" is available. It offers the same features as the driver "ABB Arcnet Route".

Setting up the Gateway channels and the parameterization is also performed in the same way as with the driver "ABB Arcnet Route".

In addition, this driver has the parameter "Block size" (128 .. 480). With this parameter, the number of user data of a block is set. As the default, 128 is entered. This is equal to the block size of the other ARCNET drivers. **Not allowed are the values 227...245.**

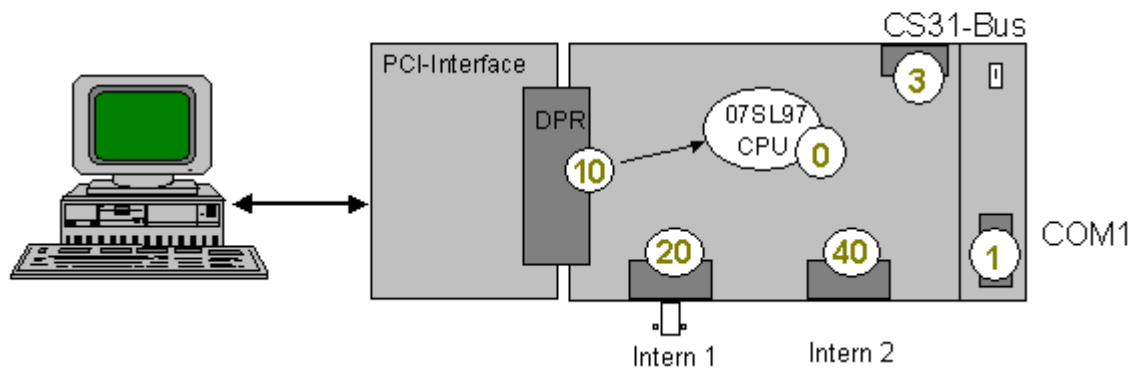


9.5 Programming 07 SL 97 slot PLC via the PCI interface

The 07 SL 97 slot PLC is accessible for programming as follows:

PC with Slot-SPS 07SL97 (PCI-Interface)

Driver: ABB SL97



Interface	Driver	Remarks
COM1	ABB RS232	as of runtime system V4.11 (system cable 07 SK 52)
	ABB RS232 Route	as of runtime system V4.16 (system cable 07 SK 52)
ARCNET	ABB Arcnet	as of runtime system V4.11
	ABB Arcnet Route	as of runtime system V4.15
	ABB Arcnet Route fast	as of runtime system V5.0
PCI	ABB SL97	as of runtime system V4.152
	ABB SL97 fast	as of runtime system V5.0

For the interfaces COM1 and ARCNET, the settings are equal to those of the 07 KT 95..98 PLCs.

9.5.1 PCI Routing driver "ABB SL97"

The installation of the PCI driver is described in Volume 6 – Installation. The PCI Routing driver is available for the following PC operating systems:

- WinNT V4.0 (Service Pack 5 or later)
- Win2000

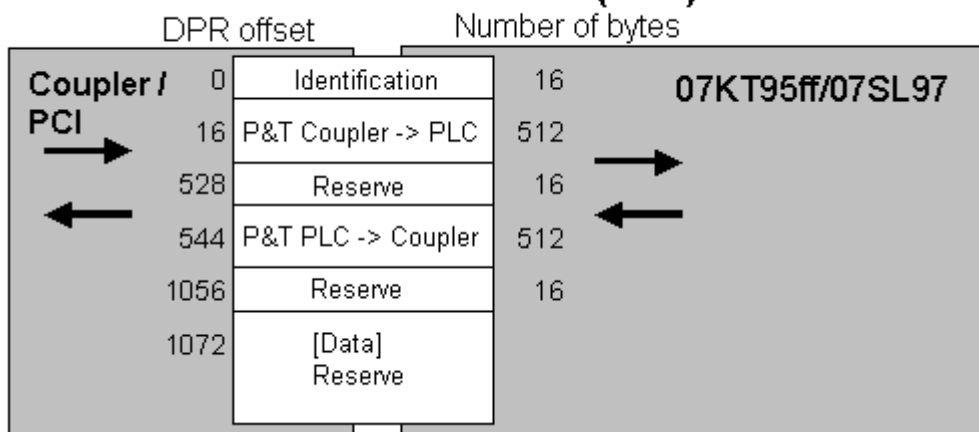
No PCI driver is available for the PC operating system Win98 SE.

The PCI Routing driver "ABB SL97" offers:

- Online operation of the 07 SL 97 slot PLC with 907 AC 1131
- Online operation of the 07 SL 97 slot PLC with 907 FB 1131
- OPC connection with OPC server as of V2.xx
- Online operation of PLCs, networked with ARCNET, via the 07 SL 97 slot PLC
- Parallel operation of 907 AC 1131 and 907 FB 1131
- Parallel operation of 907 AC 1131 and OPC server

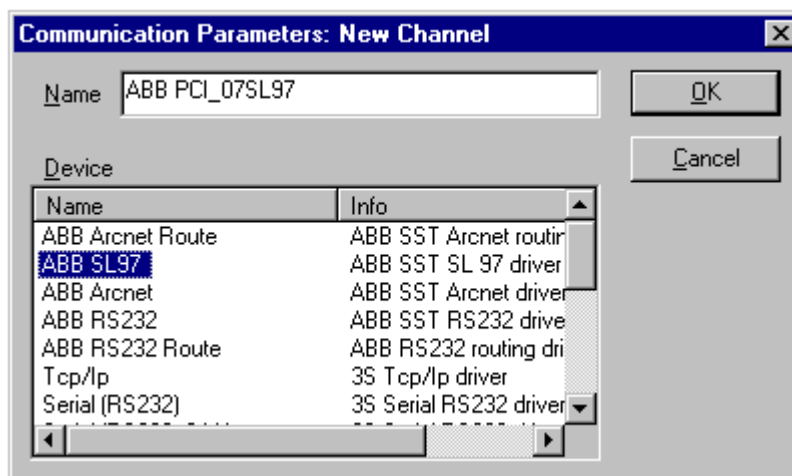
The communication between the PCI interface and the working processor is carried out via a dual-port RAM (DPR).

Construct of the Dual Port RAM (DPR)

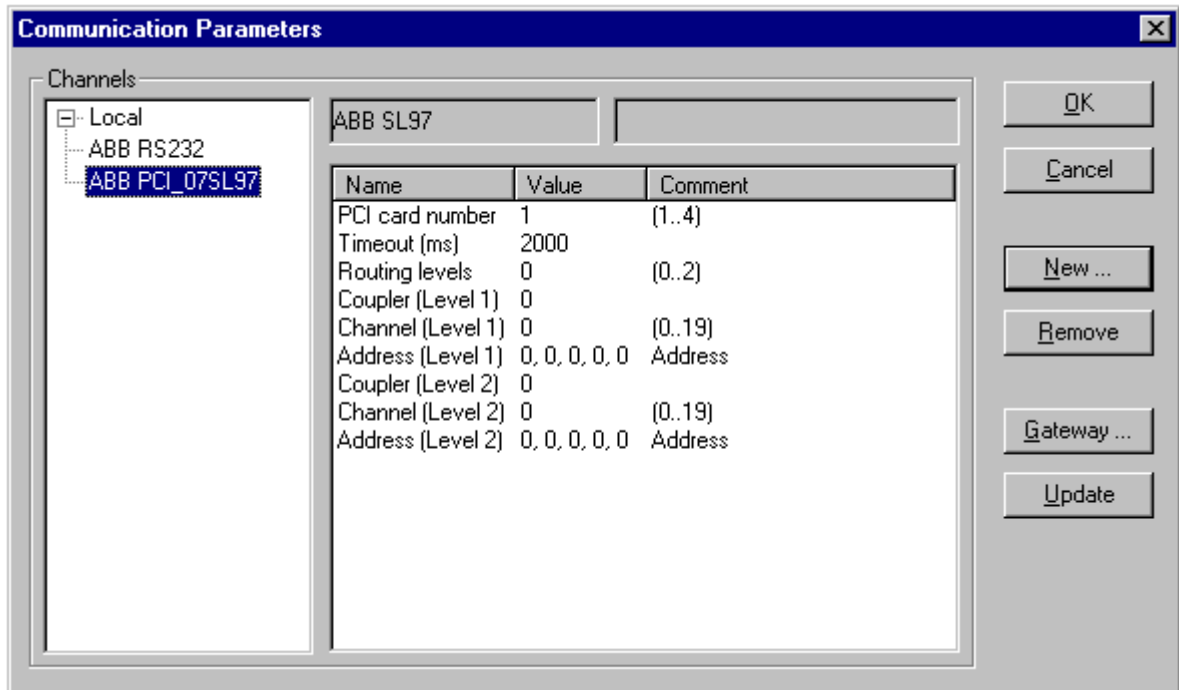


For programming, the range up to byte 1071 is reserved.

In order to establish a new Gateway channel for the PCI Routing driver, select "Online/Communication parameters/New", enter a name for the new channel (e.g. ABB PCI_07SL97) and then select the "ABB SL97" driver.



For the PCI Routing driver "ABB SL97", the following communication parameter mask is used:



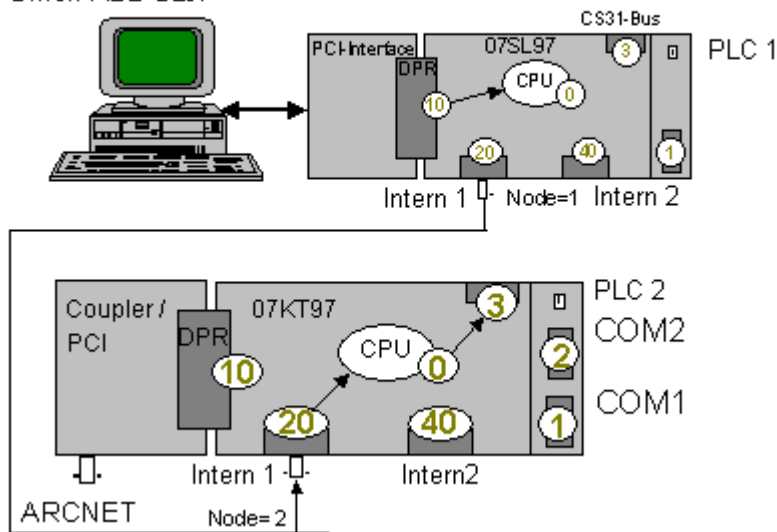
Parameter	Possible values	Meaning
PCI card number	1..4 (default: 1)	At present always 1
Timeout (ms)	>= 2000	Timeout [ms] for response
Routing levels	0..2	Routing levels (0 = none)
Coupler (Level 1)	0, Extern, Intern 1..Intern 4	Coupler for level 1
Channel (Level 1)	0..19	Channel to coupler level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 1
Coupler (Level 2)	0, Extern, Intern 1..Intern 4	Coupler for level 2
Channel (Level 2)	0..19	Channel to coupler level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 2

If it is required to directly access to the connected 07 SL 97 slot PLC with the PCI Routing driver, all of the Routing parameters (as of Routing levels) must be entered with 0.

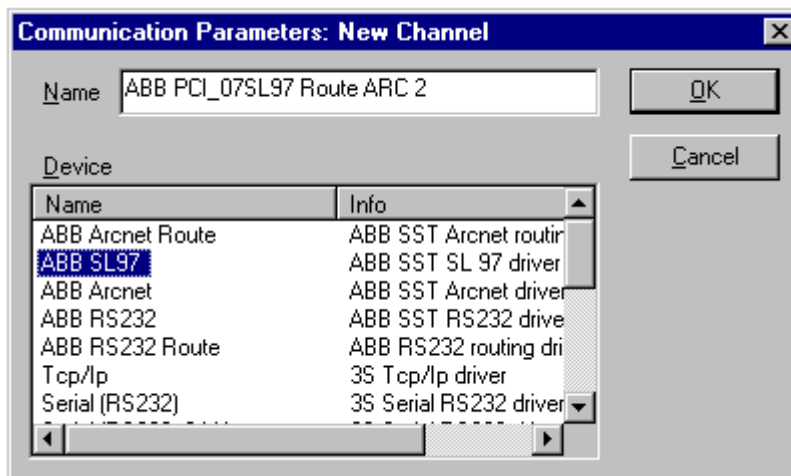
PCI ARCNET Routing:

The PLCs, networked via ARCNET, can be programmed via the PCI interface of the 07 SL 97 slot PLC using the PCI Routing driver.

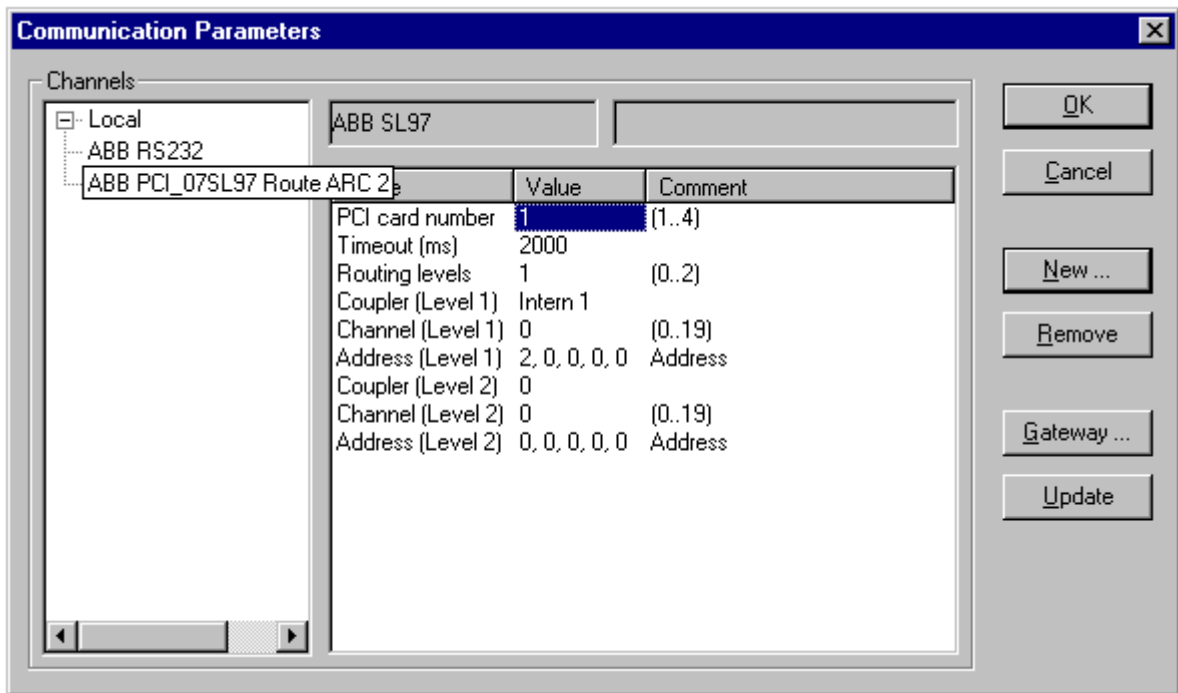
PC with 07 SL 97 and over Intern 1 (Node=1) at PLC 2 ARCNET Node=2
online PLC 2 (ARCNET Node=2)
Driver: ABB SL97



For each PLC, connected via ARCNET, a separate Gateway channel must be established. Select "Online/Communication parameters/New", enter a name for the channel (e.g. ABB PCI_07SL97 Route ARC 2) and then select the "ABB SL97" driver.



For the example, described above, you can set the following communication parameters:



Parameter	Possible values	Meaning
PCI card number	1	At present always 1
Timeout (ms)	2000	Timeout [ms] for response
Routing levels	1	One-level Routing
Coupler (Level 1)	Intern 1	Coupler for level 1 (ARCNET)
Channel (Level 1)	0	Channel to coupler level 1
Address (Level 1)	2, 0, 0, 0, 0	ARCNET node of target PLC (node 2)
Coupler (Level 2)	0	No level 2
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	

The ARCNET coupler of the 07 SL 97 slot PLC is always the coupler 1. Thus it is entered with "Intern 1".

The ARCNET coupler has only one communication channel. Therefore, "Channel" must always be entered with 0.

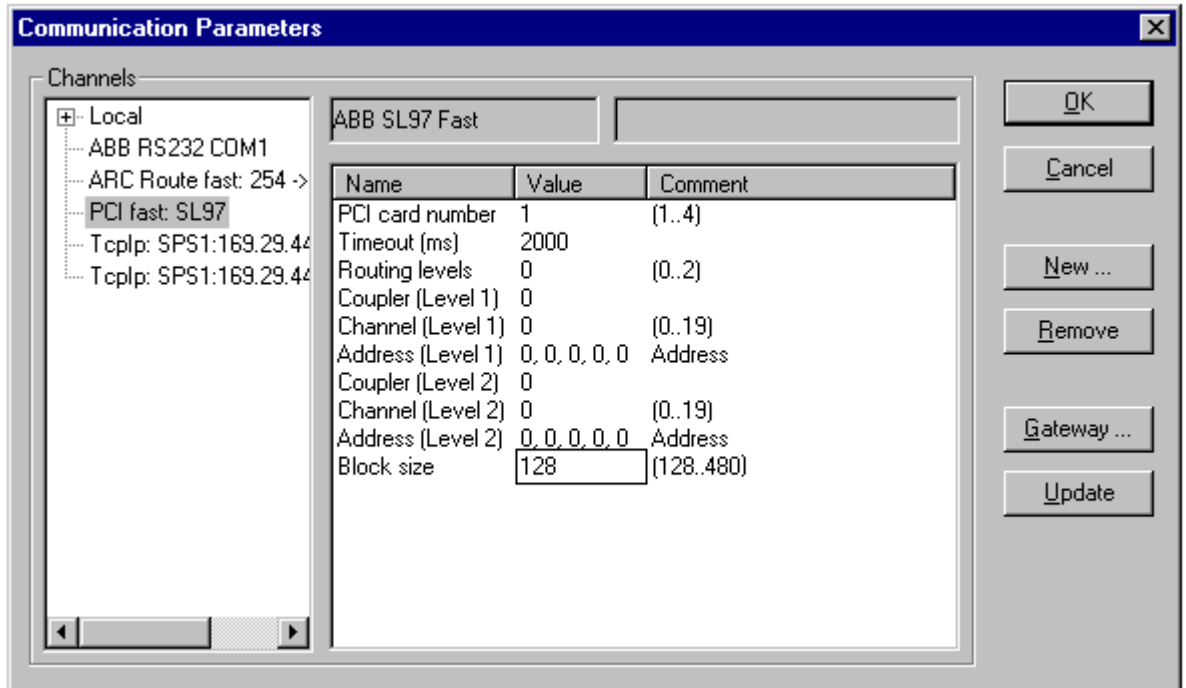
For the node address, one byte is necessary for the ARCNET coupler. The address is entered into the first byte of the address bytes.

9.5.2 PCI Routing Driver "ABB SL97 fast"

As of runtime system V5.0 of the PLC and 907 AC 1131 V5.0, the additional driver "ABB SL97 fast" is available. It offers the same features as the driver "ABB SL97".

Setting up the Gateway channels and the parameterization is also performed in the same way as with the driver "ABB SL97".

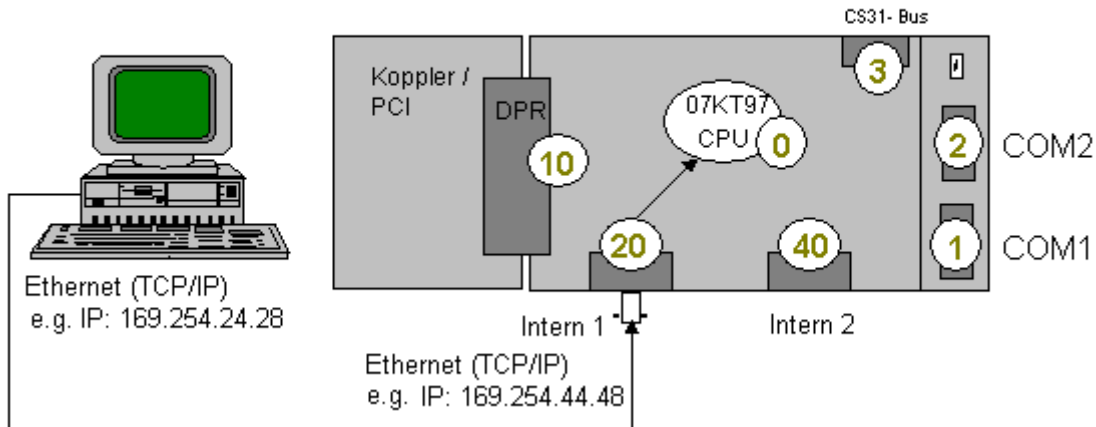
In addition, this driver has the parameter "Block size" (128 .. 480). With this parameter, the number of user data of a block is set. As the default, 128 is entered. This is equal to the block size of the other drivers. **Not allowed are the values 227...245.**



9.6 Programming via Ethernet (Tcp/Ip)

The programming via Ethernet is only possible with a PC with built-in Ethernet card and an already installed network.

PC with Ethernet (TCP/IP) / Online PLC with IP: 169.254.44.48
Driver: ABB Tcp/Ip Level 2

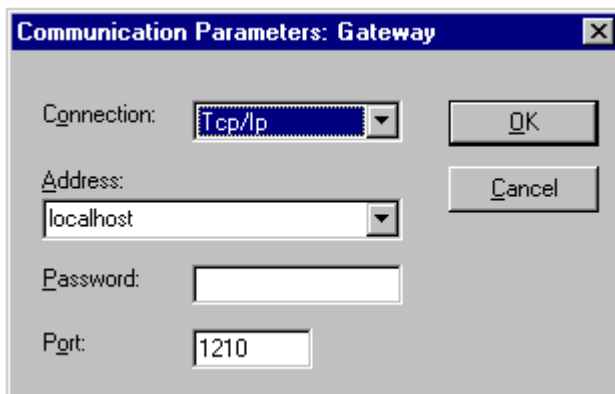


Note:

In the section "System technology"=>"System technology, internal couplers"=>"The Ethernet coupler" you can find help for setting the IP addresses.

Gateway setting for Ethernet:

Select in the programming system 907 AC 1131 under "Online/Communication parameters/Gateway" the setting "Tcp/Ip" (see Operating Manual 907 AC 1131 / Chapter 4 – The individual Components / Online/Communication Parameters with Gateway Use):

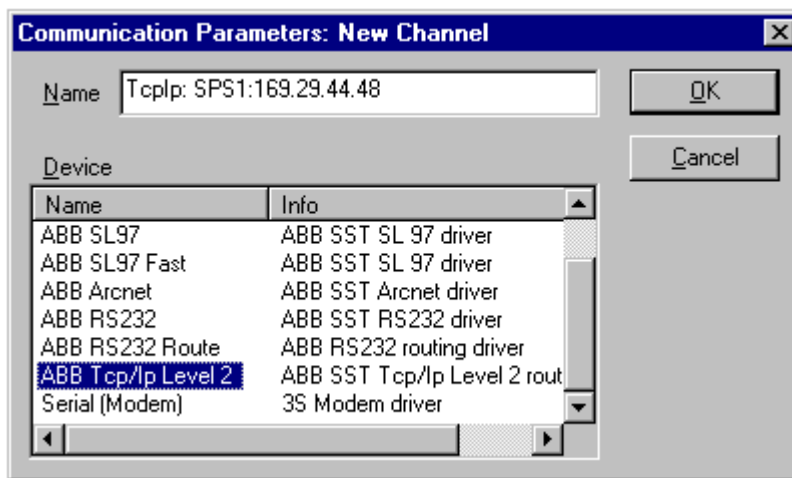


9.6.1 Ethernet driver "ABB Tcp/Ip Level 2"

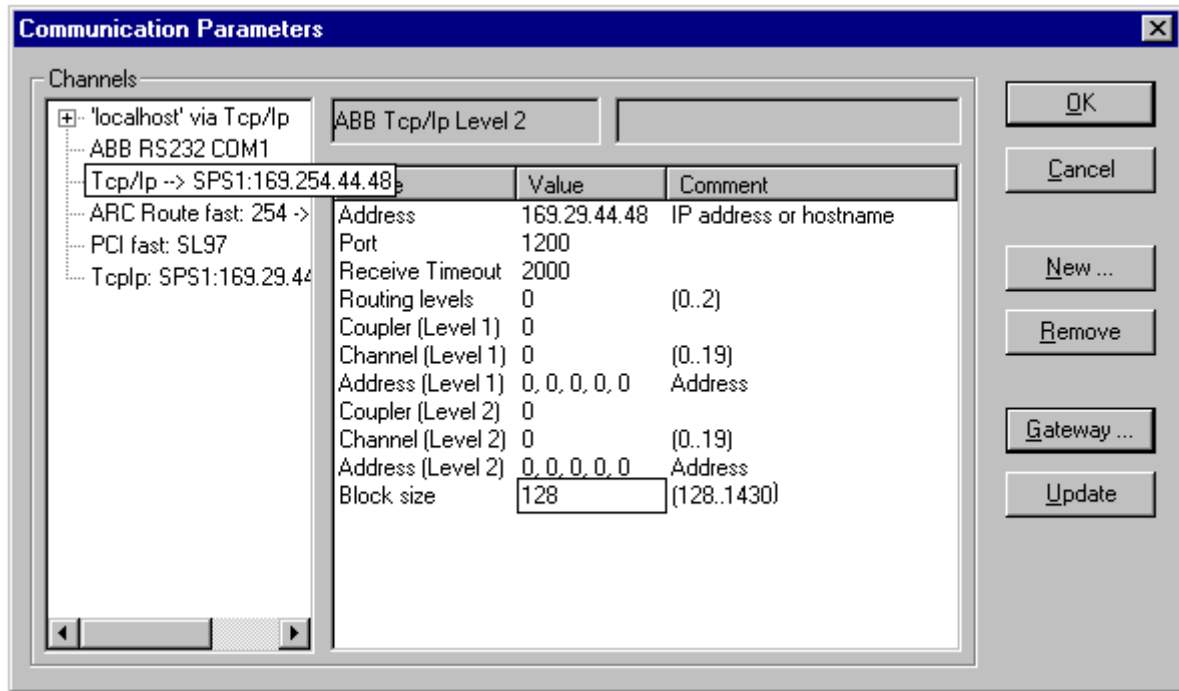
As of **runtime system V5.0** of the PLC and 907 AC 1131 V5.0, the driver "**ABB Tcp/Ip Level 2**" is available for programming control systems with integrated Ethernet couplers. This driver has the features:

- Online operation of the PLC with 907 AC 1131
- Online operation of the PLC with 907 FB 1131
- OPC connection with OPC server as of V2.xx
- Parallel operation of 907 AC 1131 and 907 FB 1131
- Parallel operation of 907 AC 1131 and OPC server
- Parallel operation of instances of AC1131.EXE with several PLCs
- Online operation of PLCs, networked via ARCNET, with a PLC with an Ethernet and an ARCNET coupler (Routing Ethernet -> ARCNET)

In order to establish a new Gateway channel for the Ethernet interface, select "Online/Communication parameters/New", enter a name for the channel (e.g. Tcp/Ip --> SPS1:169.254.44.48) and then select the driver "ABB Tcp/Ip Level 2".



For the Ethernet driver "ABB Tcp/Ip Level 2", the following communication parameter mask is valid:



Parameter	Possible values	Meaning
Address	0.0.0.0	IP address or host name of the PLC
Port	1200	
Timeout (ms)	>= 2000	Timeout [ms] for response
Routing levels	0..2	Routing steps (0 = none)
Coupler (Level 1)	0, Extern, Intern 1..Intern 4	Coupler for step 1
Channel (Level 1)	0..19	Channel on coupler step 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in the target coupler step 1
Coupler (Level 2)	0, Extern, Intern 1..Intern 4	Coupler for step 2
Channel (Level 2)	0..19	Channel on coupler step 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in the target coupler step 2
Block size	128 (128..1430)	Bytes per telegram (forbidden 227..245)

If the PLC is to be accessed directly by the Ethernet driver, all of the Routing parameters (as of Routing levels) must be set to 0.

Under "Address", the IP address or the host name of the PLC is entered. In order to make host names usable, they must be entered in the file "Hosts" using the editor. The file "Hosts" can be found, for instance, under WinNT V4.0 under "WinNT\System32\drivers\etc".

```

# Copyright (c) 1993-1995 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows NT.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com          # source server
#       38.25.63.10      x.acme.com            # x client host

127.0.0.1        localhost
169.254.44.48    SPS_1          # Maschinenteil 1
169.254.34.38    SPS_2          # Maschinenteil 2
  
```

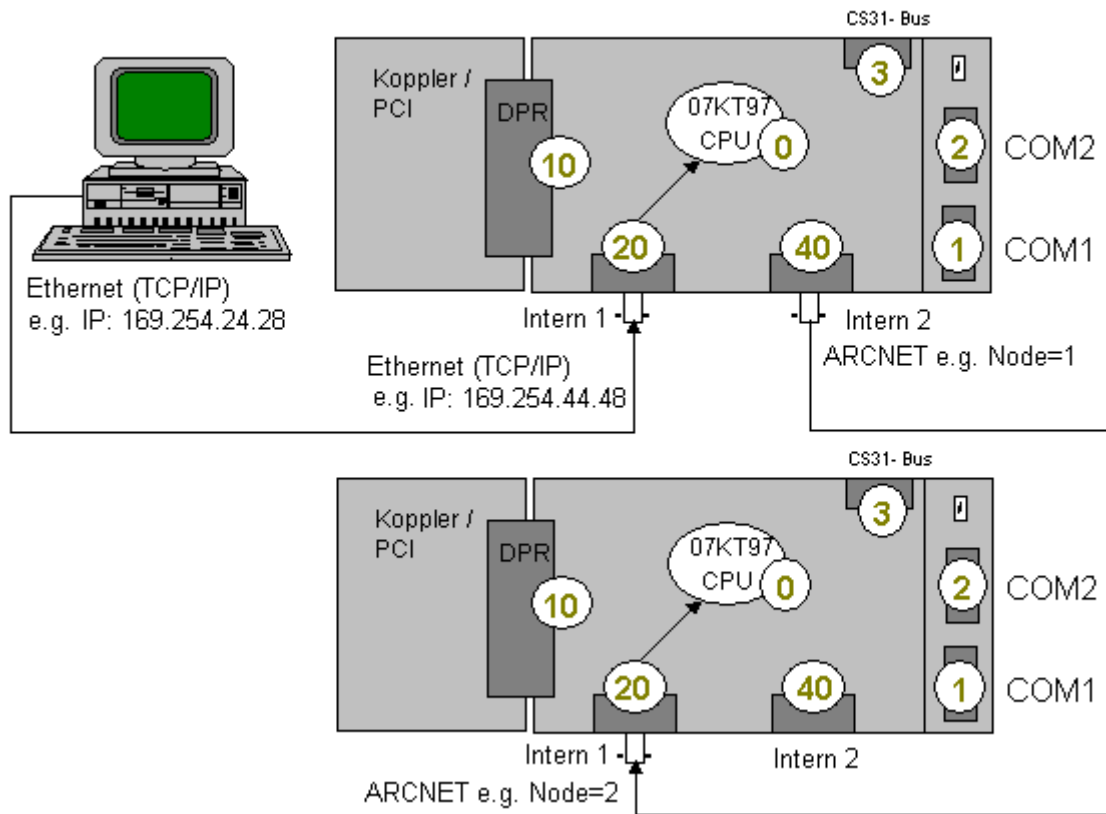
In the Communication Parameter Mask, the symbolic name is entered instead of the IP address, in the example "SPS_2" instead of "169.254.34.38":

Name	Value	Comment
Address	SPS_2	IP address or hostname
Port	1200	
Receive Timeout	2000	
Routing levels	0	(0..2)
Coupler (Level 1)	0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	0, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Block size	128	(128..1430)

Ethernet ARCNET Routing:

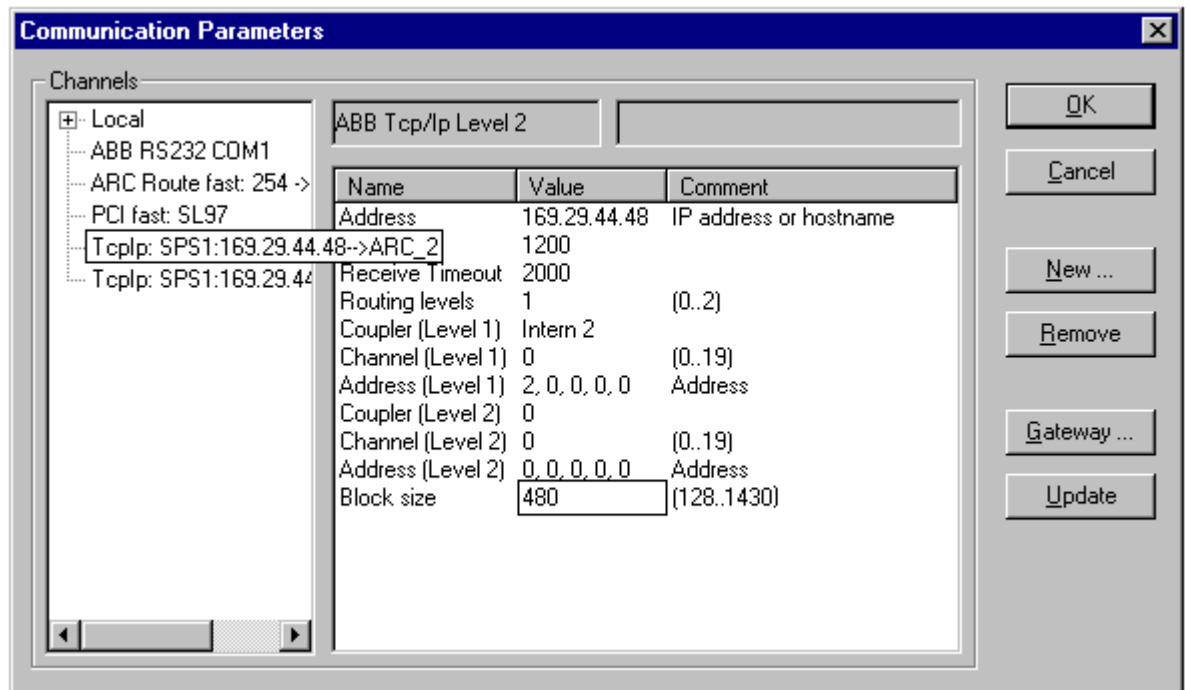
When using control systems with Ethernet and ARCNET couplers, it is possible to program PLCs, which are networked via ARCNET, via the Ethernet interface of the PLC.

PC with Ethernet (TCP/IP) / Online PLC with ARCNET Node=2
Driver: ABB Tcp/Ip Level 2



For each PLC, connected via ARCNET, a Gateway channel must be established. Select "Online/Communication parameter/New", enter a name for the channel (e.g. TcpIp: SPS1:169.29.44.48→ARC_2) and then select the driver "ABB Tcp/Ip Level 2".

For the example shown above, set the following communication parameters, for instance:



Parameter	Possible values	Meaning
Address	169.254.44.48	IP address of the PLC 1
Port	1200	
Timeout (ms)	2000	Timeout [ms] for response
Routing levels	1	One-step Routing
Coupler (Level 1)	Intern 2	Coupler for step 1 (ARCNET)
Channel (Level 1)	0	Channel on coupler step 1
Address (Level 1)	2, 0, 0, 0, 0	ARCNET node of the target PLC (node 2)
Coupler (Level 2)	0	No step 2
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	
Block size	480	Bytes per block: 128..1430

If a PLC has both an Ethernet and an ARCNET coupler, the ARCNET coupler is always the coupler 2 and thus entered with "Intern 2".

The ARCNET coupler has only one communication channel. Therefore, under "Channel" always 0 must be entered.

For the node address, 1 byte is needed for the ARCNET coupler. The address (Node=2) of the target PLC is entered to the first byte of the address bytes.

As a default, the block size 128 is entered. If it should be routed on ARCNET (and the target PLC is set to "big ARCNET packages"), the block size can be increased to 480 bytes. **The values 227 .. 245 are not allowed.**

10 List of function blocks

10.1 Overview of libraries

All libraries and their function blocks are listed in tables in the description "Overview of the libraries" of 907 AC 1131.

11 MODBUS RTU for the serial interfaces

11.1 Protocol description

The MODBUS protocol is used all over the world. The **MODICON MODBUS® RTU** protocol is integrated in the basic unit 07 KT 97.

Numerous automation devices, such as PLC installations, displays, variable-frequency inverters or monitoring systems, for instance, have a MODBUS® RTU interface by default or as an option and can therefore communicate with the basic unit 07 KT 97 without any problems via the serial interfaces COM1 and COM2 (RS-232).

MODBUS® is a master-slave protocol. The master sends a request to the slave and receives its response.

Description of the MODBUS® protocol:

- USupported standard EIA RS-232
- Number of connection points 1 master
max. 1 slave with RS-232 interface
max. 31 slaves with RS-232/RS-485 converter
- Protocol MODBUS® (master/slave)
- Data transmission control CRC 16
- Data transmission rate up to 19.200 baud
- Character encoding 1 start bit
8 data bits
1 parity bit, even or odd (optional)
1 or 2 stop bits
- Maximum cable length for RS485: 1.200 m with 19.200 baud

The MODBUS® blocks, transferred by the master, contain the following information:

- the MODBUS® address of the interrogated slave (1 byte)
- the function code which defines the request of the master (1 byte)
- the data to be exchanged (N bytes)
- the CRC16 control code (2 bytes)

The basic unit 07 KT 97 processes only the following MODBUS® operation codes:

Function codes (HEX)	Description
01 or 02	Read n bits
03 or 04	Read n words
05	Write one bit
06	Write one word
07	Fast reading of flags M01,00...M01,07 / %MX1.0..%MX1.7
0F	Write n bits
10	Write n words

The following restrictions apply to the lengths:

Function code (HEX)	Max. length for master	Max. length for slave
01 or 02	192 bits (up to runtime system V4.15) 1536 bits (as of runtime system 5.0)	192 bits (up to runtime system V4.15) 1536 Bit (as of runtime system 5.0)
03 or 04	96 words / 48 double words	96 words / 48 double words
05	1 bit	1 bit
06	1 word	1 word
07	8 bits	8 bits
0F	192 bits (up to runtime system V4.15) 1536 bits (as of runtime system 5.0)	192 bits (up to runtime system V4.15) 1536 bits (as of runtime system 5.0)
10	96 words / 48 double words	96 words / 48 double words

11.2 MODBUS operating modes of the serial interfaces COM1 and COM2

Both serial interfaces of the basic unit 07 KT 97 can be operated simultaneously as MODBUS interfaces.

The MODBUS operating mode of an interface is set by means of the function block MODINIT. The interface is selected via the block input COM. The input MAST_SLAV determines the operating mode of the selected interface:

- MODBUS master: MAST_SLV = 100
- MODBUS slave: MAST_SLV = 100 + slave address

The settings are applied together with the interface parameters when a FALSE/TRUE edge occurs at the input FREI.

MODBUS master (MAST_SLV = 100)

In the operating mode MODBUS master, the telegram interchange with the slave(s) is performed via the MODMAST function block. The function block MODMAST sends the MODBUS request telegrams to the slave via the set interface and receives the MODBUS response telegrams from the slave via this interface.

All the telegrams are to be processed via a MODMAST function block.

MODBUS slave (MAST_SLV = 100 + slave address)

In the operating mode MODBUS slave, no function block is required for the MODBUS communication. Sending and receiving of MODBUS telegrams is performed automatically.

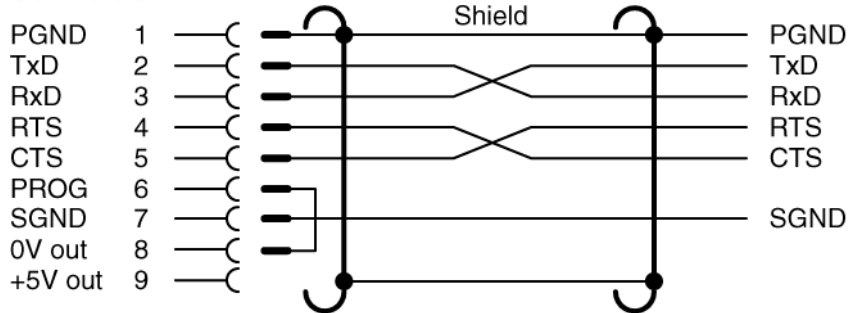
Initialization of the interfaces COM1 and COM2

The initialization of the interface is performed via the MODINIT block. The interface parameters are applied together with the selection of the interface (COM) and the presetting for the operating mode (MAST_SLV) when a FALSE/TRUE edge occurs at the input FREI.

11.3 Cable

The cable used as the data transmission line must have the pins 6 and 8 bridged in the connector on the controller side, like the interface cable 07 SK 91.

COM1 / COM2



PGND	Protective Ground	RTS	Request To Send
TxD	Transmit Data	CTS	Clear To Send
RxD	Receive Data	PROG	Switch-over active/passive mode
		SGND	Signal Ground

Circuit diagram COM1 / COM2 for MODBUS

11.4 MODBUS telegrams

The sending and receiving telegrams shown are not visible in the PLC. In both operating modes the function block MODINIT only indicates whether a telegram was received (REC = TRUE) or sent (SND = TRUE) and which function code the telegram contains (FCT). However, the complete telegrams can be displayed using a serial data analyzer which is inserted into the connection line between master and slave, if required.

The amount of user data depends on the properties of the master and the slave.

For the following examples it is assumed that a SST MODBUS module is used as slave. There may be different properties if modules of other manufacturers are used.

FCT 1 or 2: Read n bits

n = 1... 192 (up to runtime system V4.15)

n = 1...1536 (as of runtime system V5.0)

Master request

Slave address	Func. code	Op. addr. slave		No. of bits		CRC	
		High	Low	High	Low	High	Low

Slave-Response

Slave address	Func. code	No. of bytes	...data...	CRC	
				High	Low

Example: MODBUS interface of master: COM1
Master reads from: Slave 1
Data: M01,04 = FALSE; M01,05 = TRUE;
M01,06 = FALSE
Source address on slave: M01,04: 2014_{HEX} = 8212_{DEZ}
Target address on master: M10,01
The values of the flags M01,04...M01,06 on the slave are stored on the master starting with M10,01.

MODBUS request of the master

Slave address	Func. code	Op. addr. slave		No. of bits		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	01 _{HEX}	20 _{HEX}	14 _{HEX}	00 _{HEX}	03 _{HEX}	37 _{HEX}	CF _{HEX}

MODBUS response of the slave

Slave address	Func. code	No. of bytes	Data	CRC	
				High	Low
01 _{HEX}	01 _{HEX}	01 _{HEX}	02 _{HEX}	D0 _{HEX}	49 _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	1	Application-specific	8212	3	ADR (M10,01)

FCT 3 or 4: Read n words

n = 1...96

Master request

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Func. code	No. of bytes	...Data...	CRC	
				High	Low

Example: MODBUS interface of master: COM1
 Master reads from: Slave 1
 Data: MW00,04 = 4; MW00,05 = 5; MW00,06 = 6
 Source address on slave: MW00,04: 2004_{HEX} = 8196_{DEZ}
 Target address on master: MW10,01
 The values of the flag words MW00,04...MW00,06 on the slave are stored on the master starting with MW10,01.

MODBUS request of the master

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	03 _{HEX}	20 _{HEX}	04 _{HEX}	00 _{HEX}	03 _{HEX}	4F _{HEX}	CA _{HEX}

MODBUS response of the slave

Slave address	Func. code	No. of bytes	Data		Data		Data		CRC	
			High	Low	High	Low	High	Low	High	Low
01 _{HEX}	03 _{HEX}	06 _{HEX}	00 _{HEX}	04 _{HEX}	00 _{HEX}	05 _{HEX}	00 _{HEX}	06 _{HEX}	40 _{HEX}	B6 _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	3	Application-specific	8196	3	ADR (MW0,01)

FCT 3 or 4: Read n double words

The function code "Read double word" is not defined in the MODBUS RTV standard. This is why the double word is composed of a low word and a high word (manufacturer-specific).

n = 1...48

Master request

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Func. code	No. of bytes	...Data...	CRC	
				High	Low

Example: MODBUS interface of master: COM1
 Master reads from: Slave 1
 Data: MD00,02 = 32; MD00,03 = 80000
 Source address on slave: MD00,02: 4002_{HEX} = 16386_{DEZ}
 Target address on master: MD00,00
 The values of the flag double words MD00,02...MD00,03 on the slave are stored on the master starting with MD00,00.

MODBUS request of the master

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	03 _{HEX}	40 _{HEX}	02 _{HEX}	00 _{HEX}	04 _{HEX}	F0 _{HEX}	09 _{HEX}

MODBUS response of the slave

Slave address	Func. code	No. of bytes	Data		Data		Data		Data		CRC	
			High	Low	High	Low	High	Low	High	Low	High	Low
01 _{HEX}	03 _{HEX}	08 _{HEX}	00 _{HEX}	00 _{HEX}	00 _{HEX}	20 _{HEX}	00 _{HEX}	01 _{HEX}	38 _{HEX}	80 _{HEX}	57 _{HEX}	B0 _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	31	Application-specific	16386	4	ADR (MD00,00)

FCT 5: Write 1 bit

For the function code "Write 1 bit", the value of the bit to be written is encoded in one word.

BIT = TRUE → Data word = FF 00_{HEX}

BIT = FALSE → Data word = 00 00_{HEX}

n = 1

Master request

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low

Example: MODBUS interface of master: COM1
 Master writes to: Slave 1
 Data: M10,01 = TRUE
 Source address on master: M10,01
 Target address on slave: M01,04: 2017_{HEX} = 8215_{DEZ}
 The value of the flag M10,01 on the master is stored on the slave in M01,04.

MODBUS request of the master

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	05 _{HEX}	20 _{HEX}	17 _{HEX}	FF _{HEX}	00 _{HEX}	37 _{HEX}	FE _{HEX}

MODBUS response of the slave (mirrored)

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	05 _{HEX}	20 _{HEX}	17 _{HEX}	FF _{HEX}	00 _{HEX}	37 _{HEX}	FE _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	5	Application-specific	8215	1	ADR (M10,01)

FCT 6: Write 1 word

n = 1

Master request

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low

Example: MODBUS interface of master: COM1
Master writes to: Slave 1
Data: MW10,01 = 7
Source address on master: MW10,01
Target address on slave: MW00,07: 2007_{HEX} = 8199_{DEZ}
The value of the flag word MW10,01 on the master is stored on the slave in MW00,07.

MODBUS request of the master

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	06 _{HEX}	20 _{HEX}	07 _{HEX}	00 _{HEX}	07 _{HEX}	72 _{HEX}	09 _{HEX}

MODBUS response of the slave (mirrored)

Slave address	Func. code	Op. addr. slave		Data		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	06 _{HEX}	20 _{HEX}	07 _{HEX}	00 _{HEX}	07 _{HEX}	72 _{HEX}	09 _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	6	Application-specific	8215	1	ADR (MW0,01)

FCT 7: Fast reading M01,00...M01,07

Master request

Slave address	Func. code	CRC	
		High	Low

Slave response

Slave address	Func. code	Data byte	CRC	
			High	Low

Example: MODBUS interface of master: COM1
 Master writes to: Slave 1
 Data: M01,00 = TRUE; M01,01 = FALSE;
 M01,02 = TRUE; M01,03 = FALSE;
 M01,04 = TRUE; M01,05 = FALSE;
 M01,06 = TRUE; M01,07 = FALSE
 Source address on slave: M01,00: 2010_{HEX} = 8208_{DEZ}
 Target address on master: M01,08
 The values of the flags M01,00...M01,07 on the slave are stored on the master in M01,08...M01,15.

MODBUS request of the master

Slave address	Func. code	CRC	
		High	Low
01 _{HEX}	07 _{HEX}	41 _{HEX}	E2 _{HEX}

MODBUS response of the slave

Slave address	Func. code	Data byte	CRC	
			High	Low
01 _{HEX}	07 _{HEX}	55 _{HEX}	E2 _{HEX}	0F _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	7	Application-specific	8208	8	ADR (M10,01)

FCT 15: Write n bits

n = 1... 192 (if 07 KT 9X is the MODBUS slave, up to runtime system V4.15)

n = 1...1536 (if 07 KT 9X is the MODBUS slave, as of runtime system V5.0)

Master request

Slave address	Func. code	Op. addr. slave		No. of bits		No. of bytes	...Data...	CRC	
		High	Low	High	Low			High	Low

Slave response

Slave address	Func. code	Op. addr. slave		No. of bits		CRC	
		High	Low	High	Low	High	Low

Example: MODBUS interface of master: COM1
 Master writes to: Slave 1
 Data: M01,01 = TRUE; M01,02 = FALSE;
 M01,03 = TRUE
 Source address on master: M01,01
 Target address on slave: M01,01: 2011_{HEX} = 8209_{DEZ}
 The values of the flags M01,01...M01,03 on the master are stored on the slave in M01,01...M01,03.

MODBUS request of the master

Slave address	Func. code	Op. addr. slave		No. of bits		No. of bytes	Data	CRC	
		High	Low	High	Low			High	Low
01 _{HEX}	0F _{HEX}	20 _{HEX}	11 _{HEX}	00 _{HEX}	03 _{HEX}	01 _{HEX}	05 _{HEX}	B4 _{HEX}	37 _{HEX}

MODBUS response of the slave

Slave address	Func. code	Op. addr. slave		No. of bits		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	0F _{HEX}	20 _{HEX}	11 _{HEX}	00 _{HEX}	03 _{HEX}	4E _{HEX}	0F _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	15	Application-specific	8209	3	ADR (M01,01)

FCT 16: Write n words

n = 1...96 (if 07 KT 9X is the MODBUS slave)

Master request

Slave address	Fkt.-code	Op. addr. slave		No. of words		No. of bytes	...Data...	CRC	
		High	Low	High	Low			High	Low

Slave response

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low

Example: MODBUS interface of master: COM1
 Master writes to: Slave 1
 Data: MW01,01 = 1; MW01,02 = 2; MW01,03 = 3
 Source address on master: MW01,01
 Target address on slave: MW00,01: 2001_{HEX} = 8193_{DEZ}
 The values of the flag words MW01,01...MW01,03 on the master are stored on the slave in MW00,01...MW00,03.

MODBUS request of the master

Slave addr.	Func. code	Op. addr. slv.		No. of words		No. of bytes	Data		Data		Data		CRC	
		High	Low	High	Low		High	Low	High	Low	High	Low	High	Low
01 _{HEX}	10 _{HEX}	20 _{HEX}	01 _{HEX}	00 _{HEX}	03 _{HEX}	06 _{HEX}	00 _{HEX}	01 _{HEX}	00 _{HEX}	02 _{HEX}	00 _{HEX}	03 _{HEX}	C0 _{HEX}	84 _{HEX}

MODBUS response of the slave

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	10 _{HEX}	20 _{HEX}	01 _{HEX}	00 _{HEX}	03 _{HEX}	DA _{HEX}	08 _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	16	Application-specific	8193	3	ADR (MW01,01)

FCT 16: Write n double words

The function code "Write double word" is not defined in the MODBUS RTV standard. This is why the double word is composed of a low word and a high word (manufacturer-specific).

n = 1...48

Master request

Slave address	Func. code	Op. addr. slave		No. of words		No. of bytes	...Data...	CRC	
		High	Low	High	Low			High	Low

Slave response

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low

Example: MODBUS interface of master: COM1
 Master writes to: Slave 1
 Data: MD00,00 = 18; MD00,01 = 65561
 Source address on master: MD00,00
 Target address on slave: MD00,00: 4000_{HEX} = 16384_{DEZ}
 The values of the flag double words MD00,00...MD00,01 on the master are stored on the slave in MD00,00...MD00,01.

MODBUS request of the master

Slave addr.	Func. code	Op. addr. slv.		No. of words		No. of bytes	Data		Data		Data		Data		CRC	
		High	Low	High	Low		High	Low	High	Low	High	Low	High	Low		
01 _{HEX}	10 _{HEX}	40 _{HEX}	00 _{HEX}	00 _{HEX}	04 _{HEX}	08 _{HEX}	00 _{HEX}	00 _{HEX}	00 _{HEX}	12 _{HEX}	00 _{HEX}	01 _{HEX}	00 _{HEX}	19 _{HEX}	60 _{HEX}	B3 _{HEX}

MODBUS response of the slave

Slave address	Func. code	Op. addr. slave		No. of words		CRC	
		High	Low	High	Low	High	Low
01 _{HEX}	10 _{HEX}	40 _{HEX}	00 _{HEX}	00 _{HEX}	04 _{HEX}	DA _{HEX}	0A _{HEX}

Parameter setting for the block inputs MODMAST

NB = Number of words = 2 x Number of double words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE ->TRUE	1	1	16	Application-specific	16384	4	ADR (MD00,00)

Error telegram

In the operating mode MODBUS master, the 07 KT 97 sends a telegram only if the parameters at the MODMAST inputs are logically correct. Nevertheless it can happen that a slave cannot process the request of the master or that the slave cannot interpret the request due to transmission errors. In those cases the slave sends an error telegram to the master. As an identification for the error telegram the slave sends back a function code resulting of a logical OR interconnection of the function code received from the master and the value 80_{HEX}.

Slave response

Slave address	Function code OR 80 _{HEX}	Error code	CRC	
			High	Low

Possible error code of the slave

Code Meaning

- 01_{DEC} The slave does not support the function requested by the master
- 02_{DEC} Invalid operand address in the slave
- 02_{DEC} Operand area exceeded
- 03_{DEC} At least one value is outside the permitted value range
- 12_{DEC} The amount of data is higher than the slave can process
- 13_{DEC} The telegram contains an odd number of words in case of double word access
- 10_{DEC} The length specifications in the telegram do not match
- 11_{DEC} The type of operand area and the function do not match

Example:

MODBUS request of the master:

Function code: 01 (Read n bits)
Operand address slave: 4000_{HEX} = 16384_{DEZ} (flag area double word)

MODBUS response of the slave

Function code: 81_{HEX}
Error code: 02

11.5 Function block MODINIT

This function block initializes and configures a serial interface to be a MODBUS interface. It is required for the operating mode MODBUS master as well as for the operating mode MODBUS slave. This function block can be applied to the local controller interfaces (COM1/COM2) and to the interfaces of the MODBUS coupler 07 KP 93 (COM3/COM4). A separate instance of the function block must be used for each interface.

MODMAST is part of the libraries **ABB-BIB4.LIB** and **COM_S90_V4x.LIB** (as of V4.1).

11.6 Function block MODMAST

This function block is only required in the operating mode MODBUS master. It deals with the communication (sending telegrams to the slaves and receiving telegrams from the slaves). This function block can be applied to the local controller interfaces (COM1/COM2) and to the interfaces of the MODBUS coupler 07 KP 93 (COM3/COM4). A separate instance of the function block must be used for each interface.

MODMAST is part of the libraries **ABB-BIB4.LIB** and **COM_S90_V4x.LIB** (as of V4.1).

11.7 Cross-reference list



Caution:

An access to reserved operands (e.g. E065_00 = %IX065.00) or undefined operands (e.g. E64_08 = %IX064.08) is not permissible and leads to unforeseeable results. This applies to the operand area in the master (MODMAST input DATA) as well as to the operand area in the slave (MODMAST input ADDR).

Please note that the forbidden operands can also be reached by a valid start address and a length NB (e.g. DATA = ADR(E064_00) and NB = 8 or ADDR = 400_{HEX} and NB = 8). Such combinations of addresses and lengths must be absolutely avoided.



Caution:

Writing accesses to the inputs must be avoided.

When writing via MODBUS to inputs of a 07 KT 97 which acts as a MODBUS slave, the written values are immediately overwritten by the "true" input values. When the 07 KT 97 acting as MODBUS master reads via MODBUS and the data are stored to inputs, the values which were read are also immediately overwritten again by the "true" input values.



Caution:

When writing via MODBUS to constants of the 07 KT 97 which acts as a MODBUS slave, the written values are first only applied to the running operation. When the 07 KT 97 acting as MODBUS master reads via MODBUS and the data are stored to constants, the values which were read are also first only applied to the running operation.

In order to overwrite constants permanently, they must be additionally saved in the Flash memory. Otherwise the original values of the project are loaded after the supply voltage is switched on once again.

Operands (symbolic)	Operands (IEC)	MODBUS address (HEX)	Operand description
E000_00 ⋮ E061_15	%IX000.00 ⋮ %IX061.15	0000 _{HEX} ⋮ 03DF _{HEX}	Digital inputs, CS31 bus
E062_00 ⋮ E063_15	%IX062.00 ⋮ %IX063.15	03E0 _{HEX} ⋮ 03FF _{HEX}	Digital inputs, local
E064_00 ⋮ E064_07	%IX064.00 ⋮ %IX064.07	0400 _{HEX} ⋮ 0407 _{HEX}	Digital inputs, local (binary access to EW006_00..EW006_07)
E065_00 ⋮ E099_15	%IX065.00 ⋮ %IX099.15	0410 _{HEX} ⋮ 063F _{HEX}	Digital inputs, central expansion (reserved)
E100_00 ⋮ E163_15	%IX100.00 ⋮ %IX163.15	0640 _{HEX} ⋮ 0A3F _{HEX}	Digital inputs, 2nd decentralized expansion (reserved)
E200_00 ⋮ E255_15	%IX200.00 ⋮ %IX255.15	0C80 _{HEX} ⋮ 0FFF _{HEX}	Digital inputs, 3rd decentralized expansion (reserved)
E256_00 ⋮ E263_15	%IX256.00 ⋮ %IX263.15	No direct access	Digital inputs, 3rd decentralized expansion (reserved)
	%IX1.1.0000.0 ⋮ %IX1.1792.15	No direct access	Digital inputs, PROFIBUS line 1
	%IX2.1.0000.0 ⋮ %IX2.1792.15	No direct access	Digital inputs, PROFIBUS line 2

Operands (symbolic)	Operands (IEC)	MODBUS address (HEX)	Operand description
A000_00 ⋮ A061_15	%QX000.00 ⋮ %QX061.15	1000 _{HEX} ⋮ 13DF _{HEX}	Digital outputs, CS31 bus
A062_00 ⋮ A063_15	%QX062.00 ⋮ %QX063.15	13E0 _{HEX} ⋮ 13FF _{HEX}	Digital outputs, local
A064_00 ⋮ A099_15	%QX064.00 ⋮ %QX099.15	1400 _{HEX} ⋮ 163F _{HEX}	Digital outputs, central expansion (reserved)
A100_00 ⋮ A163_15	%QX100.00 ⋮ %QX163.15	1640 _{HEX} ⋮ 1A3F _{HEX}	Digital outputs, 2nd decentralized expansion (reserved)
A200_00 ⋮ A255_15	%QX200.00 ⋮ %QX255.15	1C80 _{HEX} ⋮ 1FFF _{HEX}	Digital outputs, 3rd decentralized expansion (reserved)
A256_00 ⋮ A263_15	%QX256.00 ⋮ %QX263.15	no direct access	Digital outputs, 3rd decentralized expansion (reserved)
	%QX1.1.0000.0 ⋮ %QX1.1792.15	no direct access	Digital outputs, PROFIBUS line 1
	%QX2.1.0000.0 ⋮ %QX2.1792.15	no direct access	Digital outputs, PROFIBUS line 2

Operands (symbolic)	Operands (IEC)	MODBUS address (HEX)	Operand description
M000_00 ⋮ M254_15	%MX0000.00 ⋮ %MX0254.15	2000 _{HEX} ⋮ 2FEF _{HEX}	Binary flags
M255_00 ⋮ M255_15	%MX0255.00 ⋮ %MX0255.15	2FF0 _{HEX} ⋮ 2FFF _{HEX}	Binary flags (system)
M256_00 ⋮ M279_15	%MX0256.00 ⋮ %MX0279.15	no direct access	Binary flags (system, reserved)
M280_00 ⋮ M511_15	%MX0256.00 ⋮ %MX0511.15	no direct access	Binary flags (system)
S000_00 ⋮ S255_15	%MX5000.00 ⋮ %MX5255.15	3000 _{HEX} ⋮ 3FFF _{HEX}	Steps

Operands (symbolic)	Operands (IEC)	MODBUS address (HEX)	Operand description
EW000_00 ⋮ EW005_15	%IW1000.00 ⋮ %IW1005.15	0000 _{HEX} ⋮ 005F _{HEX}	Analog inputs, CS31 bus
EW006_00 ⋮ EW006_07	%IW1006.00 ⋮ %IW1006.07	0060 _{HEX} ⋮ 0067 _{HEX}	Analog inputs, local
EW007_00 ⋮ EW007_14	%IW1007.00 ⋮ %IW1007.14	0070 _{HEX} ⋮ 007E _{HEX}	Analog inputs (reserved)
EW007_15	%IW1007.15	007F _{HEX}	Analog inputs, Status word CS31 bus
EW008_00 ⋮ EW015_15	%IW1008.00 ⋮ %IW1015.15	0080 _{HEX} ⋮ 00FF _{HEX}	Analog inputs, CS31 bus
EW016_00 ⋮ EW034_15	%IW1016.00 ⋮ %IW1034.15	0100 _{HEX} ⋮ 022F _{HEX}	Analog inputs, central expansion (reserved)
EW100_00 ⋮ EW107_15	%IW1100.00 ⋮ %IW1107.15	0640 _{HEX} ⋮ 06BF _{HEX}	Analog inputs, 1st decentralized expansion (reserved)
EW200_00 ⋮ EW207_15	%IW1200.00 ⋮ %IW1207.15	0C80 _{HEX} ⋮ 0CFF _{HEX}	Analog inputs, 2nd decentralized expansion (reserved)
	%IW1.0000 ⋮ %IW1.1792	no direct access	Analog inputs, PROFIBUS line 1
	%IW2.0000 ⋮ %IW2.1792	no direct access	Analog inputs, PROFIBUS line 2

Operands (symbolic)	Operands (IEC)	MODBUS address (HEX)	Operand description
AW000_00 ⋮ AW005_15	%QW1000.00 ⋮ %QW1005.15	1000 _{HEX} ⋮ 105F _{HEX}	Analog outputs, CS31 bus
AW006_00 ⋮ AW006_03	%QW1006.00 ⋮ %QW1006.03	1060 _{HEX} ⋮ 1063 _{HEX}	Analog outputs, local
AW007_00 ⋮ AW007_15	%QW1007.00 ⋮ %QW1007.15	1070 _{HEX} ⋮ 107F _{HEX}	Analog outputs (reserved)
AW008_00 ⋮ AW015_15	%QW1008.00 ⋮ %QW1015.15	1080 _{HEX} ⋮ 10FF _{HEX}	Analog outputs, CS31 bus
AW016_00 ⋮ AW034_15	%QW1016.00 ⋮ %QW1034.15	1100 _{HEX} ⋮ 122F _{HEX}	Analog outputs, central expansion (reserved)
AW100_00 ⋮ AW107_15	%QW1100.00 ⋮ %QW1107.15	1640 _{HEX} ⋮ 16BF _{HEX}	Analog outputs, 1st decentralized expansion (reserved)
AW200_00 ⋮ AW207_15	%QW1200.00 ⋮ %QW1207.15	1C80 _{HEX} ⋮ 1CFF _{HEX}	Analog outputs, 2nd decentralized expansion (reserved)
	%QW1.0000 ⋮ %QW1.1792	no direct access	Analog outputs, PROFIBUS line 1
	%QW2.0000 ⋮ %QW2.1792	no direct access	Analog outputs, PROFIBUS line 2

Operands (symbolic)	Operands (IEC)	MODBUS address (HEX)	Operand description
MW000_00 ⋮ MW253_15	%MW1000.00 ⋮ %MW1253.15	2000 _{HEX} ⋮ 2FDF _{HEX}	Word flags
MW254_00 ⋮ MW255_15	%MW1254.00 ⋮ %MW1255.15	2FE0 _{HEX} ⋮ 2FFF _{HEX}	Word flags (error message)
MW256_00 ⋮ MW259_15	%MX1256.00 ⋮ %MX1259.15	no direct access	Word flags (system, reserved)
MW260_00 ⋮ MW511_15	%MW1260.00 ⋮ %MW1511.15	no direct access	Word flags (user area)
KW000_00 ⋮ KW000_15	%MW3000.00 ⋮ %MW3000.15	3000 _{HEX} ⋮ 300F _{HEX}	Word constants (system)
KW001_00 ⋮ KW079_15	%MW3001.00 ⋮ %MW3079.15	3010 _{HEX} ⋮ 34FF _{HEX}	Word constants
KW080_00 ⋮ KW089_15	%MW3080.00 ⋮ %MW3089.15	3500 _{HEX} ⋮ 359F _{HEX}	Word constants (system)
MD000_00 ⋮ MD063_15	%MD2000.00 ⋮ %MD2063.15	4000 _{HEX} ⋮ 43FF _{HEX}	Double word flags
KD000_00	%MD4000.00	5000 _{HEX}	Double word constant (system)
KD000_01 ⋮ KD023_15	%MD4000.01 ⋮ %MD4023.15	5001 _{HEX} ⋮ 517F _{HEX}	Double word constants

11.8 MODBUS example program

07 KT 97 configured with COM1 as MODBUS master and COM2 as MODBUS slave

Task: The bits %M0.1...%MX0.8 at the slave (07 KT 97 COM2) are to be read with a maximum frequency and stored in %MX0.9...%MX0.15 at the master (07 KT 97 COM1).

Note: No serial programming interface is available during the running operation because both serial interfaces of the basic unit are simultaneously operated as MODBUS interfaces. The online operation of the 907 AC 1131 is only possible via ARCNET. Alternatively the program can be loaded into two separate controllers, each having only one serial interface operated as a MODBUS interface. In this case the online access of the 907 AC 1131 can also be performed via the other serial interface which is not used as MODBUS interface.

In the program header various variables are first declared and initialized. This way of implementation increases the clarity of the program and additionally simplifies the incorporation of changes possibly required at a later time. Alternatively the values can also be declared directly at the block inputs.

```
PROGRAM MODBUSEXAMPLE
```

```
VAR
```

```
MODINIT_EN:      ARRAY[1..2] OF BOOL:= TRUE, TRUE;  (*Release MODINIT*)
MODMAST_EN:      ARRAY[1..2] OF BOOL:= FALSE, FALSE; (*Release MODMAST*)
MODBUS_IDENT:    ARRAY[1..2] OF INT  := 100, 101;    (*COM1 master, COM2 slave with Addr. 1*)
MODBUS_BAUD:     INT                  := 9600;       (*9600 baud*)
MODBUS_PTY:      INT                  := 0;          (*no parity*)
MODBUS_STOP:     INT                  := 1;          (*1 stop bit*)
MODBUS_RTSCTRL:  BOOL                 := FALSE;      (*no RTS control*)
MODBUS_TLS:      INT                  := 0;          (*0 ms carrier lead time*)
MODBUS_CDLY:     INT                  := 0;          (*0 ms carrier delay time*)
MODBUS_CHTO:     INT                  := 3;          (*3 ms character timeout*)
MODBUS_TIMEOUT: INT                  := 1000;        (*1 s telegram timeout*)
MODBUS_FCT:      INT                  := 1;          (*function "read n bits"*)
MODBUS_ADDR:     INT                  := 16#2001;    (*read starting from %MX0.1*)
MODBUS_NB:       INT                  := 7;          (*7 bits*)
MODBUS_DATA:     DWORD;               (*memory address for bits read*)

MODINIT_ERROR:   ARRAY[1..2] OF INT;                (*storage of errors*)
MODMAST_ERROR:   ARRAY[1..2] OF INT;                (*storage of errors*)

MODINIT_COM1:    MODINIT;                  (*instance*)
MODINIT_COM2:    MODINIT;                  (*instance*)
MODMAST_COM1:    MODMAST;                  (*instance*)
```

```
END_VAR
```

Example for a variable declaration and initialization

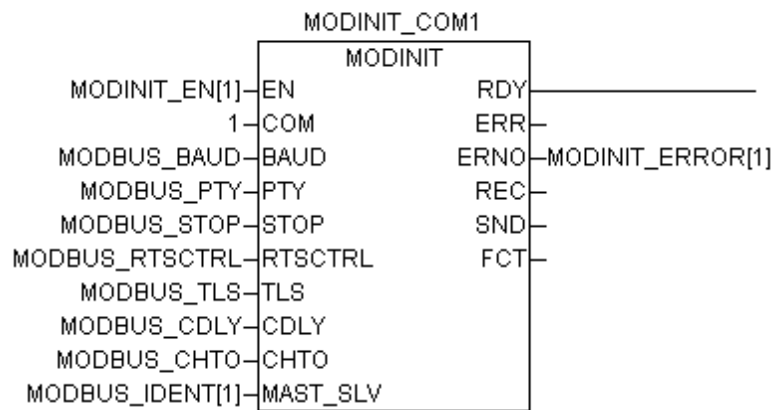
The initialization of the operand address in the master (MODBUS_DATA) is performed in the instruction part of the program by means of the ADR operator.



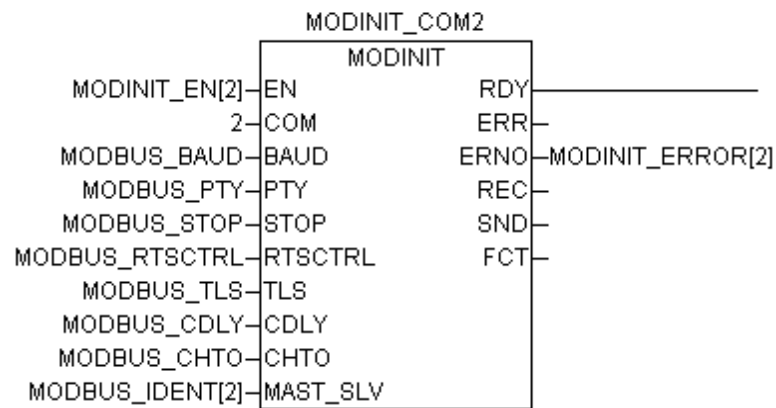
Pre-assignment of the operand address for the operating mode MODBUS master

The initialization of the serial interface is performed via instances of the MODINIT block with the help of the variables defined in the program header. A communication via the serial interfaces is only possible, if both interfaces are set to the same transmission parameters (BAUD – CHTO). The EN inputs of the MODINIT type instances are internally pre-initialized with FALSE. Consequently an initialization of the MODINIT_EN variables with TRUE causes a FALSE→TRUE edge at the EN input of the instances during the first program cycle and a permanent value of TRUE during the following cycles. This causes an initialization of the interfaces during the first cycle and an operation as MODBUS interfaces during the following cycles.

Initialization COM1



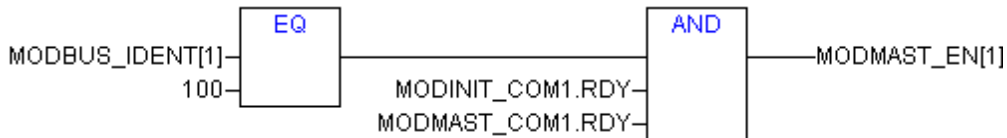
Initialization COM2



Initialization of the MODBUS interfaces

In the next step the MODBUS master functionality for COM1 must be realized with the help of the MODMAST block. In case of a FALSE → TRUE edge at the input EN, the master sends a request telegram to a slave which corresponds to the remaining block inputs. The RDY output of the MODMAST block is pre-initialized with TRUE. During processing of the telegram RDY is FALSE. RDY becomes TRUE after the processing was successful or when an error occurs. Therefore the RDY signal can be directly used to activate the block via the input EN. Because sending of telegrams is only possible after the corresponding interface was successfully initialized to be a MODBUS master interface, it is sensible to additionally interconnect the release with the RDY output of the associated MIDINIT block.

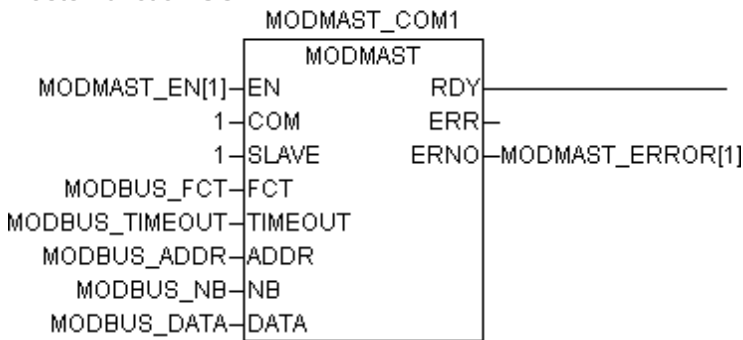
Release of master function COM1



Enabling signal for the MODMAST block

The MODMAST block is now activated as follows:

Master function COM1



Realization of the MODBUS master functionality

This example program does not consider the processing of possible errors. The current error messages are only stored to the ARRAYs MODINIT_ERROR and MODMAST_ERROR. For real projects, an additional application specific error reaction should be programmed.



Caution:

If the block is activated as described in the example program mentioned above, no jobs are executed anymore if at least one input of the MODMAST block is invalid. In order to initiate a new job, the MODMAST block requires a FALSE→TRUE edge at the release input EN. If the values at the inputs are correct, the block sets RDY = FALSE and executes the job. After the job is processed, the block sets RDY = TRUE and in case of an error it sets ERR = TRUE. Using the activation shown above, this generates a rising edge at the input EN. But if one of the inputs experiences an error, the RDY output stays TRUE and ERR is set. Thus, no new edge is generated at the input EN. See also the description of the block MODMAST.

%

%IW1007.15 2-33, 7-10
 %MW1254.0-%MW1255.15 2-32
 %MW1259.0 2-33
 %MW1259.1-%MW1259.3 2-33
 %MW3000.0 2-12
 %MW3000.1 2-13
 %MW3000.10 2-15
 %MW3000.11 2-15
 %MW3000.12 2-16
 %MW3000.15 2-16
 %MW3000.2 2-13
 %MW3000.3 2-13
 %MW3000.4 2-14
 %MW3000.7 2-14
 %MW3080.0 2-17
 %MW3080.1 2-17
 %MW3081.8 2-18
 %MW3082.0 2-18
 %MW3085.0 2-20
 %MW3085.1 2-20
 %MW3085.2 2-21
 %MW3086.x 2-22
 %MW3088.x 2-28
 %MX255.0-%MX255.6 2-31
 %MX255.10-%MX255.14 2-32
 %MX255.15 2-32
 %MX255.9 2-32
 %MX258.0-%MX256.6 2-31

A

Absolut addresses of operands 2-7
 Acknowledgement of errors 7-10
 Acknowledgement of errors at AC31 remote modules 7-7
 Address setting analog AC31 modules 5-5
 Address setting digital AC31 modules 5-4
 Analog AC31 modules 5-11
 Analog inputs 2-2
 Analog outputs 2-4
 Automatically initiated warm start after FK2 error 2-16

B

Backup of data areas S90 3-5
 Behaviour of the serial interfaces 90 series 8-2
 Buffering of data areas S90 3-2
 Bus master basic unit on the CS31 bus 5-5

C

Capacity utilization 2-33
 Categorization AC31 modules 5-11
 Changing PLC application mode 2-12

Cold start S90 3-1
 Communication drivers 90 series 9-3
 Configuration analog inputs 2-22
 Configuration basic units 2-10
 Configuration inputs E62,00-E62,15 2-20
 Configuration inputs E63,00-E63,15 2-20
 Configuration of the analog outputs 2-28
 Configuration of the oscillators 2-16
 Configuration operating mode high-speed counter 2-21
 CS31 / Bus master 5-3
 CS31 status word 2-33, 7-10
 Cycle time 4-1

D

Declaration examples for inputs 2-3
 Diagnosis 7-1
 Digital AC31 modules 5-11
 Digital inputs 2-2
 Digital outputs 2-4
 Double word constants 2-6
 Double word flags 2-6
 Download S90 3-2
 Driver ABB Arcnet 90 Series 9-10
 Driver ABB Arcnet Route 90 Series 9-11
 Driver ABB Arcnet Route fast 90 Series 9-13
 Driver ABB RS232 Route, 90 series 9-6
 Driver ABB RS-232, 90 series 9-5
 Driver ABB SL97 fast 90 series 9-19
 Driver ABB SL97, 90 series 9-15
 Driver ABB Tcp/Ip Level 2, 90 series 9-21

E

Error classes 7-1, 7-9
 Error messages from the PLC 2-32
 Errors on AC31 remote modules 7-5
 EW 07,15 2-33, 7-10
 Example declaration outputs 2-5

F

Fatal errors - FK1 7-13
 Features of the serial interfaces 90 series 8-1
 FK1 7-13
 FK2 7-14
 FK3 7-15
 FK4 7-17
 Flags 2-6

G

Gateway setting 90 series 9-3
 Gateway setting for Ethernet 90 series 9-20

I

I/O configuration at AC31 modules 6-1
Import file for operands 2-1, 2-3, 2-5, 2-10, 2-30
Indication of exceeded PLC cycle time 2-33
Initialization of binary flags 2-13
Initialization of data areas S90 3-5
Initialization of double word flags 2-13
Initialization of step chains 2-14
Initialization of word flags 2-13
Initiating a cold start S90 3-1
Initiating a warm start S90 3-1
Inputs Line 2 2-3
Inputs Lini 1 2-3
Interface parameters 90 series 8-6
Interrupt external networking interface 2-32

K

KW 00,00 2-12
KW 00,01 2-13
KW 00,02 2-13
KW 00,03 2-13
KW 00,04 2-14
KW 00,07 2-14
KW 00,10 2-15
KW 00,11 2-15
KW 00,12 2-16
KW 00,15 2-16
KW 80,00 2-17
KW 80,01 2-17
KW 81,08 2-18
KW 82,00 2-18
KW 85,00 2-20
KW 85,01 2-20
KW 85,02 2-21
KW 86,0x 2-22
KW 88,0x 2-28

L

LED displays 7-3, 7-4, 7-8
Light errors – FK3 7-15

M

M255,0-M255,6 2-31
M255,10-M255,14 2-32
M255,15 2-32
M255,9 2-32
M256,00-M256,6 2-31
Measuring ranges of the analog input channels 2-23
Measuring ranges of the analog outputs 2-28
MODBUS example program 90 series 11-20
MODBUS RTU 90 series 11-1
MODBUS RTU cable 90 series 11-3
MODBUS RTU Protocol description 11-1
MODBUS RTU telegrams 90 series 11-3
Module examples AC31 modules 5-11
Monitoring of the tasks 2-17, 2-33
MW254,00-MW255,15 2-32
MW259,00 2-33

MW259,01-MW259,03 2-33

O

Online Change S90 3-2
Operands of the basic units 2-1
Operating modes of the serial interfaces 90 series 8-1
Oscillators 2-31
Outputs Line 1 2-4
Outputs Line 2 2-4
Overview error flags 7-9
Overview System constants 2-10

P

Peculiarities Address operator 2-7
Performing and reading the I/O configuration 6-1
PLC application mode 2-12
PLC reaction to FK3 errors 2-14
Processing times 4-1
Program processing time 4-1
Programming of the Slot PLC 07 SL 97 9-14
Programming via ARCNET 90 series 9-9
Programming via Ethernet 90 series 9-20
Programming via the serial interfaces 90 series 9-4

R

Reaction time 4-1
Recommended module addresses on the CS31 system bus 5-3
Restart detection 2-32
RUN→STOP S90 3-1

S

Serial interfaces 90 series 8-1
Serious errors – FK2 7-14
Setting ARCNET timeout S90 2-18
Size receiving area of slave PLC 2-15
Size transmitting area of slave PLC 2-15
Slave basic unit on the CS31 bus 5-6
Special modules on the CS31 bus 5-11
Stand-alone basic unit 5-5
Start of the user program S90 3-3
START→STOP S90 3-1
Steps 2-6
STOP→RUN S90 3-1
STOP→START S90 3-1
Structure of addresses for AC31 modules 5-2
System constants 2-10
System flag words 2-30
System flags 2-30

W

Warm start S90 3-1
Warning - FK4 7-17
Word constants 2-6
Word flags 2-6



ABB STOTZ-KONTAKT GmbH

Eppelheimer Straße 82 Postfach 101680
69123 Heidelberg 69006 Heidelberg
Germany Germany

Telephone +49 6221 701-0
Telefax +49 6221 701-240
E-Mail desst.help@de.abb.com
Internet <http://www.abb.de/stotz-kontakt>

System Technology 90 Series Internal Couplers

In order to easier find the different sections, their page numbers begin with the letters A to L.

Description	Page number begins with
The ARCNET coupler	A
The PROFIBUS-DP coupler	B
The DeviceNet coupler	C
The Interbus-Master coupler	D
The CANopen coupler	E
The Ethernet coupler	F
Error message of the internal couplers	K
Importing 907 FB 1131 configuration data into a 907 AC 1131 project	L

Content

1	The ARCNET coupler	A 1-1
1.1	Brief overview	1-1
1.1.1	Short description and field of application	1-1
1.1.2	Data in short	1-1
1.2	Connection and transfer media	1-2
1.2.1	Attachment plug	1-2
1.2.2	Setting the participant address	1-2
1.2.3	Transfer media	1-3
1.3	Possibilities for networking	1-4
1.3.1	Linear ARCNET	1-4
1.3.2	Conversion from coaxial cable to optical fibre	1-5
1.3.3	Linear ARCNET, expanded by active distributors (active hubs)	1-6
1.3.4	Stations with an additional coupler	1-7
1.4	ARCNET implementation	1-8
1.5	Example of a program structure for the data transfer from station to station	1-9
2	Figures	2-1
3	Index	I

1.2 Connection and transfer media

1.2.1 Attachment plug

The ARCNET coupler is designed as a bus with BNC connectors for coaxial cables. The ARCNET bus is earthed inside the module by a capacitor. For EMC suppression and for protection against dangerous contact voltages, the bus has to be earthed directly at a central place.

The 07 KT 97 is connected to the ARCNET network by means of a BNC T-connector on plug X4.

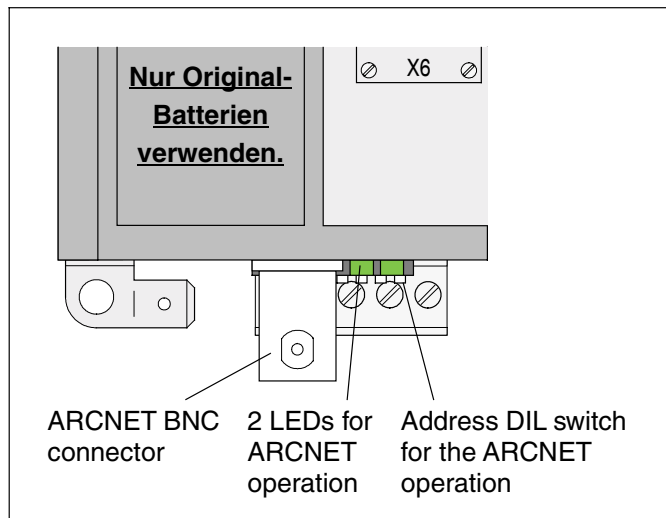


Figure 1-1 ARCNET connector

1.2.2 Setting the participant address

The DIL switch for setting the participant address is located below the plug X4. The address setting is performed binary. The following allocation applies:

$$\text{DIL 8} = 2^0 = 1$$

$$\text{DIL 7} = 2^1 = 2$$

$$\text{DIL 6} = 2^2 = 4$$

:

$$\text{DIL 1} = 2^7 = 128$$

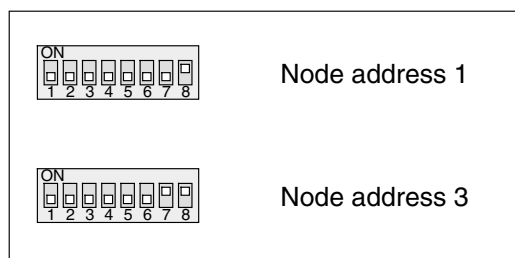


Figure 1-2 DIL switch for setting the participant address

1.2.3 Transfer media

Bus cable:

Cable RG 62 A/U:
e.g. Lapp Kabel, Stuttgart (Germany)
Phone: +49 711 7838-0

Bus cable plugs:

BNC connector 75 Ω : Order number B-9005
BNC T-connector 75 Ω : Order number B-9083
BNC terminator 93 Ω : Order number B-9093

Rufenach Vertriebs-GmbH, Heidelberg (Germany)
Phone: +49 6221 8443-0
Fax: +49 6221 8443-99

Repeaters, active star couplers:

Supplier:
APEX Automatisierungstechnik GmbH, Braunschweig (Germany)
Phone: +49 531 37040

Coaxial repeater (2 port amplifier):
010214005/HKXKX

Active star coupler, consisting of:
Modular 8 port amplifier (basic unit with controller and power supply):
010214001/modH8P

Coaxial port plug-in unit for star connection (LAND):
010214002/MHKXP

Coaxial port plug-in unit for bus connection (HIT):
010214003/MHKXP

Optical fibre converter, plug:

Refer to "Conversion from coaxial cable to optical fibre" in section 1.3.2

1.3 Possibilities for networking

Linear ARCNET is the simplest configuration. Here, the coaxial cable (RG-62 A/U, 93 Ω) is laid from one station to the next. The connection to the individual stations is done using a BNC T-connector. The ends of the cabling must be terminated by terminating resistors of 93 Ω .

1.3.1 Linear ARCNET

Linear ARCNET connects the individual stations directly to each other, without inserted distributors. The individual stations are connected to the network via T-connectors. Terminating resistors must be connected to both ends of the cabling. A maximum of 8 stations can be connected to the network. The maximum length of the network is 300 m. By means of an active hub (active distributor), an additional segment can be connected to the end of a linearly cabled segment.

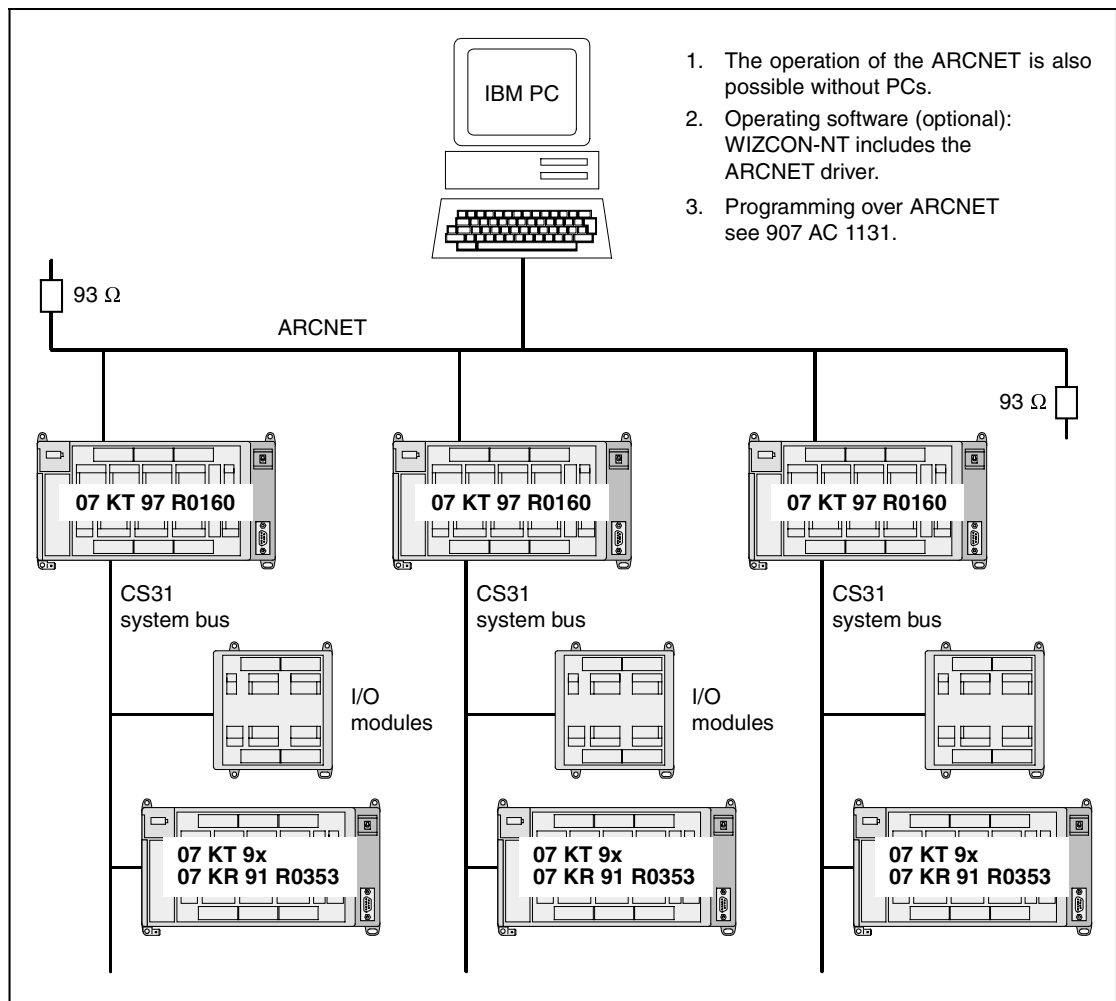


Figure 1-3 Linear ARCNET

1.3.2 Conversion from coaxial cable to optical fibre

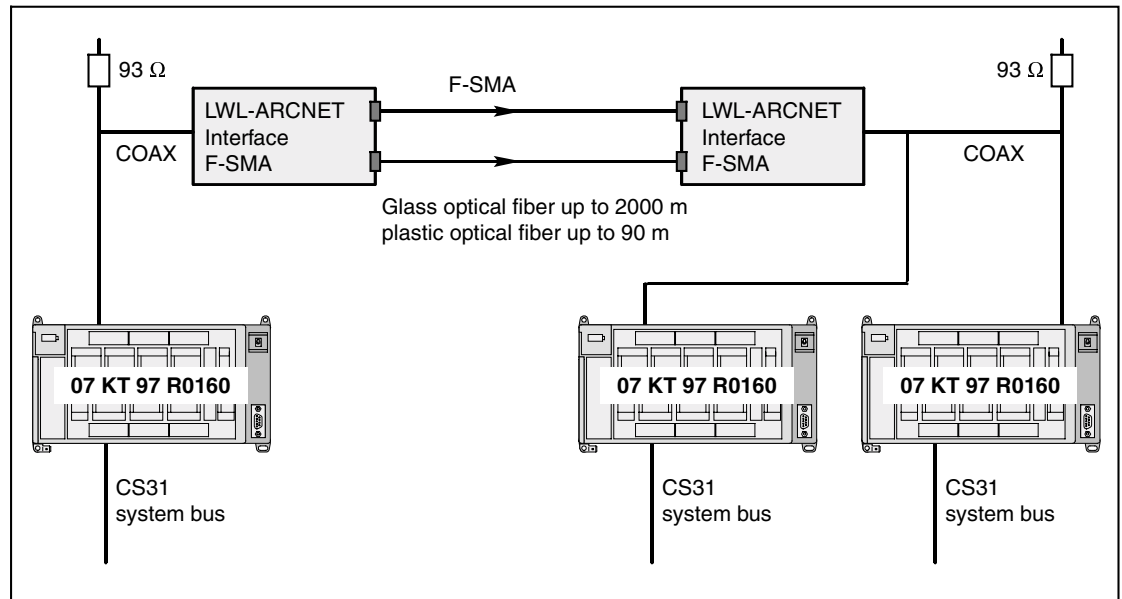


Figure 1-4 Conversion from coaxial cable to optical fibre

Supplier of optical fibre parts:

Harting Elektronik GmbH
D-32339 Espelkamp
Phone: +49 5772 47-263
Fax: + 49 5772 47-461

LWL-ARCNET interface for glass fibre:
Order number 20 40 002 3711

F-SMA connector 50/125:
Order number 20 10 125 1212

F-SMA connector 50/125 glass:
Order number 20 20 050 1022

Other cables, for example for underground wiring (on request).

For technical questions concerning optical fibre equipment, please consult Harting (phone: +49 5772 47-225).

Connecting optical fibres requires practice as a craftsman. Company for connecting optical fibres:

Magronic in Munich (Germany)
Phone: +49 89 3838-650

1.3.3 Linear ARCNET, expanded by active distributors (active hubs)

Active hubs amplify the incoming signals and therefore stabilize the network and enable longer distances in the network. The active hub uncouples the individual connections. As a result, the failure of one connection does not lead to a complete failure of the entire network. The maximum length of the network is 6.5 km. A maximum of 255 stations can be connected to the network.

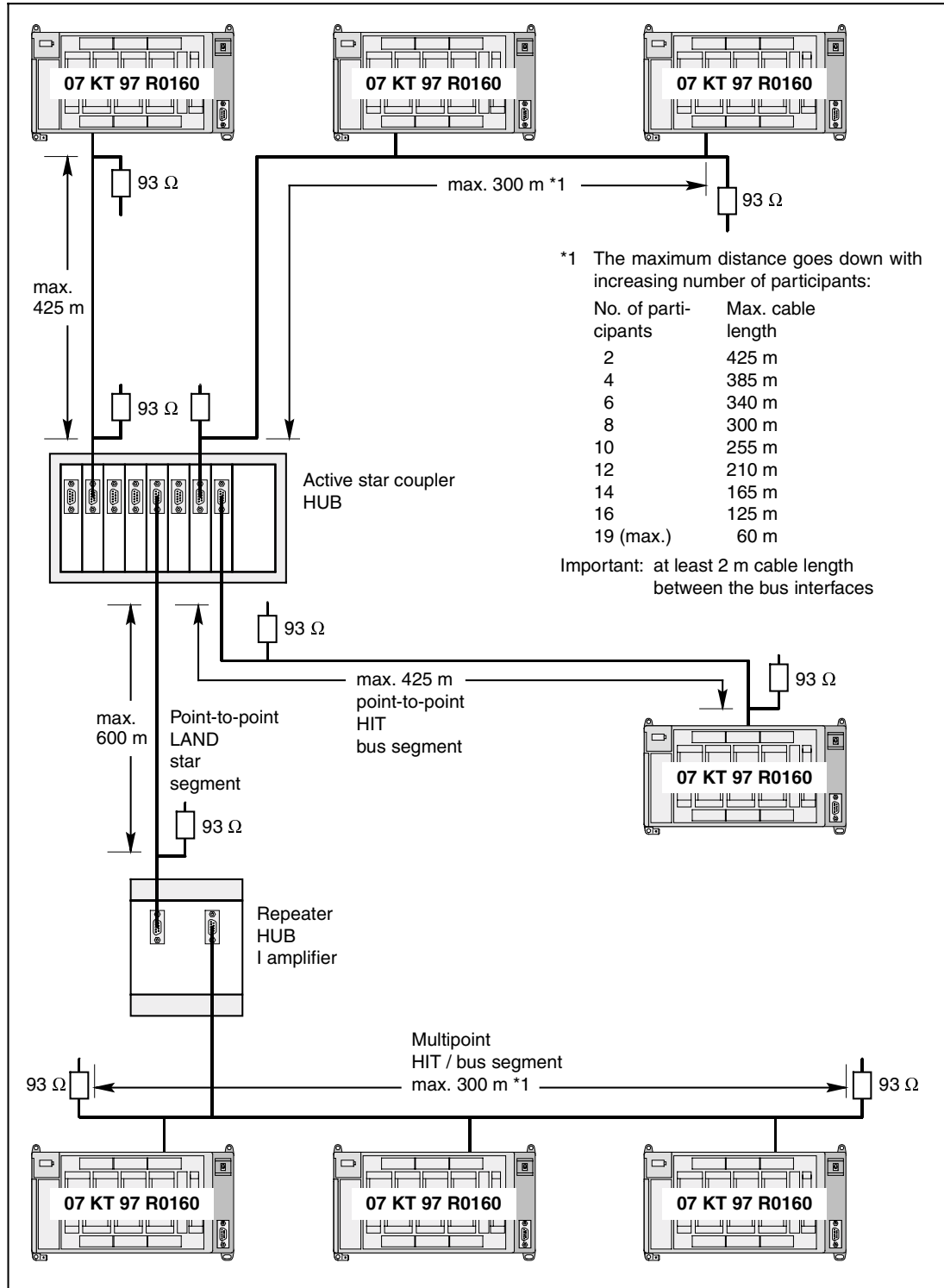


Figure 1-5 Linear ARCNET, expanded by active distributors (active hubs)

1.3.4 Stations with an additional coupler

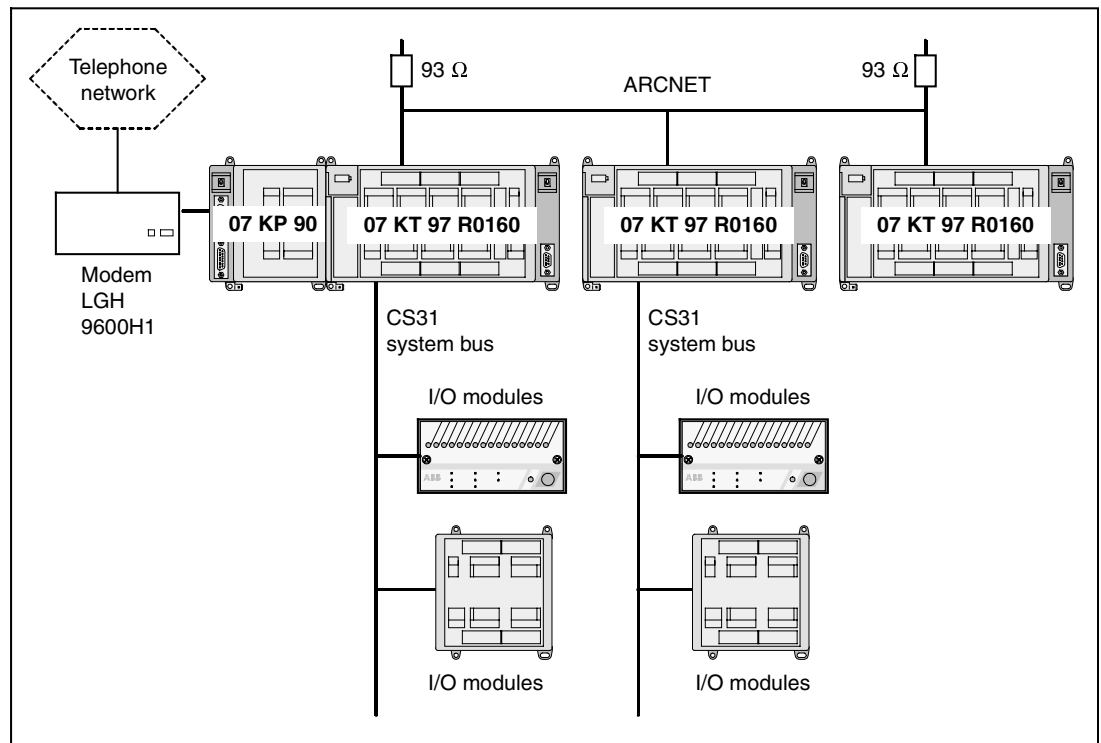


Figure 1-6 Telephone network connection for 3 substations

1.4 ARCNET implementation

The operating system has three protocols implemented which distinguish by a DIN identifier in the 1st byte of the user data:

- Programming and test protocol, DIN identifier 4F_{hex} and 3F_{hex}
- MMC protocol, DIN identifier 5F_{hex}
- Data transfer from station to station, DIN identifier 7F_{hex}

The protocols for MMC and programming of the PLC are directly handled by the operating system. Both protocols initiate a response telegram to the sender. The address of the sender is retrieved from the received telegram.

The protocol for the data transfer from station to station must be planned by the PLC user. For this purpose, the function blocks AINIT, ASEND1, ASEND4, ASEND16 and AREC are available.

The coupler is initialized exclusively for sending and receiving of short data packages. Following this, a data package for the data transfer from station to station has the following format:

- 3 bytes of check data: sender, receiver, CP
- max. 253 bytes of user data, consisting of:
1 byte DIN identifier + 2 bytes job no. + max. 125 words user data

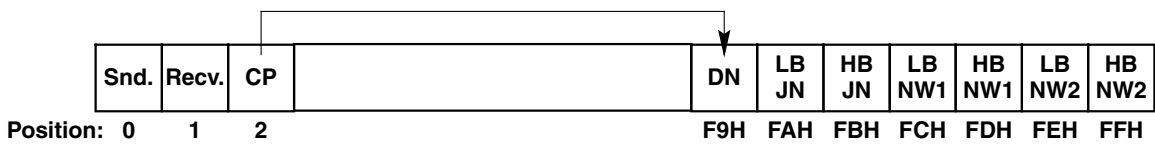


Figure 1-7 Format of a data package

Explanation:	Snd.	Station address of the sender
	Recv.	Station address of the receiver
	CP	Continuous Pointer; contains the position (F9 _H) of the 1 st byte of user data
	DN	DIN identifier (7F _H)
	LB JN	Low byte of the job number
	HB JN	High byte of the job number
	LB NW1	Low byte of user data word 1
	HB NW1	High byte of user data word 1
	LB NW2	Low byte of user data word 2
	HB NW2	High byte of user data word 2

Reading the data package from the ARCNET coupler (data reception) is performed interrupt controlled. The interrupt routine stores the data package in the receiving buffer. The AREC block reads the receiving buffer and supplies the data in the user memory. The connection elements ASEND1, ASEND4 or ASEND16 serve for the sending direction. All the connection elements store the data packages in the sending buffer. From here, the data are transferred interrupt controlled. The connection element ASEND1 sends a data package to another station. The connection element ASEND4 sends a data package to up to 4 other stations simultaneously. The connection element ASEND16 sends a data package to up to 16 other stations simultaneously.

1.5 Example of a program structure for the data transfer from station to station

As mentioned above, this data transfer protocol occurs exclusively with the DIN identifier 7FH.



Caution:

In each station just one AINIT connection element is required. All planned connection elements AINIT, AREC, ASEND1, ASEND4 and ASEND16 are to be planned in one task. It must be ensured that all implemented AREC connection elements are processed during every cycle and always in the same order. The sending connection elements ASEND1, ASEND4 and ASEND16 require a correctly planned receiving connection element AREC in an other station for the transfer (see also the descriptions of the connection elements AINIT, AREC, ASEND1, ASEND4, ASEND16). The station address for the ARCNET operation is set on a DIL switch.

Below a planning example for station addresses 1 and 2 is shown. Station 1 is sending to station 2 and to the addresses 12, 23 and 81. Station 2 is sending to station 1 and also to the addresses 12, 23 and 81. This example makes clear that the receiving connection element AREC must have the same DIN identifier and job number as it is entered at the corresponding ASEND4 connection element. The data length is not checked on the reception.

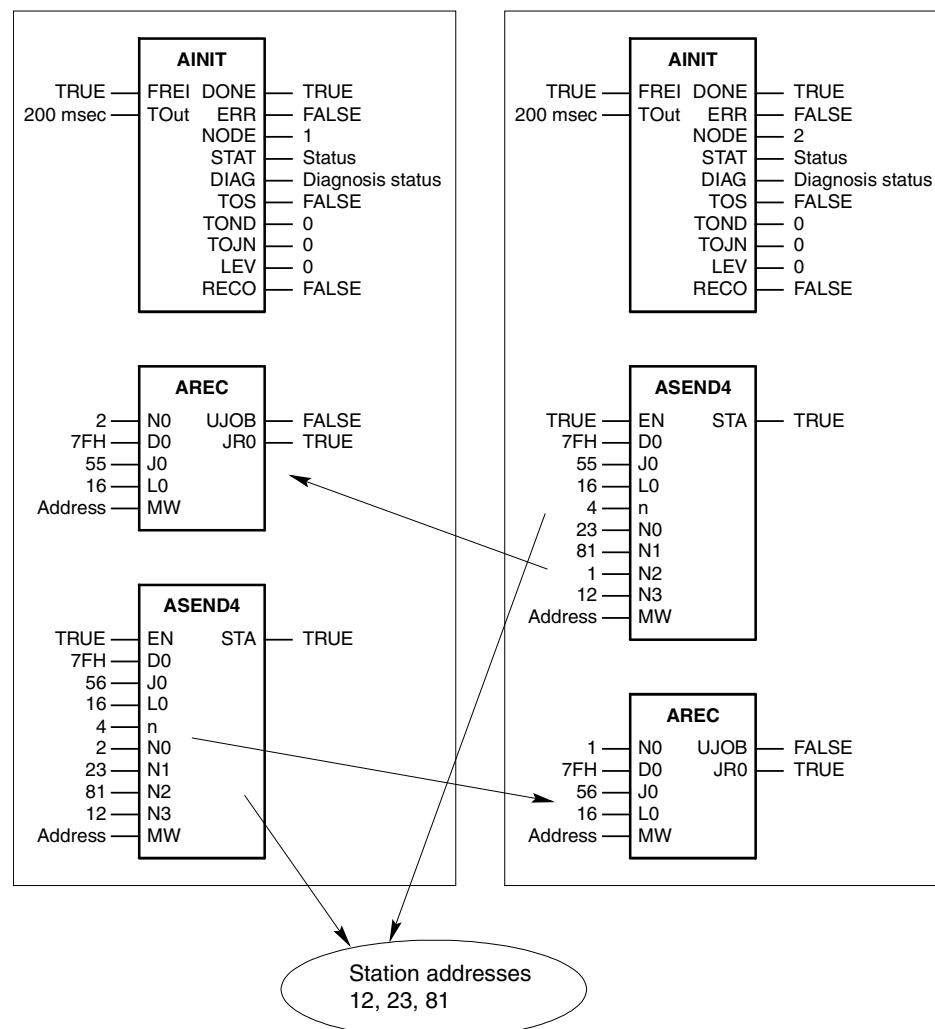


Figure 1-8 Planning example

2 Figures

Figure 1-1 ARCNET connector	1-2
Figure 1-2 DIL switch for setting the participant address	1-2
Figure 1-3 Linear ARCNET	1-4
Figure 1-4 Conversion from coaxial cable to optical fibre	1-5
Figure 1-5 Linear ARCNET, expanded by active distributors (active hubs)	1-6
Figure 1-6 Telephone network connection for 3 substations	1-7
Figure 1-7 Format of a data package	1-8
Figure 1-8 Planning example	1-9

A

ARCNET implementation 1-8

B

Brief overview ARCNET coupler 1-1

C

Connection and transfer media ARCNET 1-2

P

Possibilities for networking ARCNET 1-4

Contents

1	The PROFIBUS-DP coupler	B 1-1
1.1	Brief overview	1-1
1.1.1	Fundamental properties and fields of application	1-1
1.1.2	Features	1-2
1.2	Technical data	1-3
1.2.1	Technical data of the coupler	1-3
1.2.2	Interface specification	1-3
1.3	Connection and transfer media	1-4
1.3.1	Attachment plug for the bus cable	1-4
1.3.2	Bus terminating resistors	1-5
1.3.3	Bus cable	1-5
1.3.4	Maximum line lengths (bus segment)	1-6
1.3.5	Repeaters	1-6
1.4	Possibilities for networking	1-7
1.4.1	Single master system	1-7
1.4.2	Multi-master system	1-8
1.5	PROFIBUS implementation	1-10
1.5.1	Configuration	1-10
1.5.2	Running operation	1-11
1.5.3	Operation mode DP slave	1-11
1.5.4	Operation mode DP master	1-11
1.5.5	Error diagnosis	1-12
1.5.6	Function blocks	1-12
1.6	Planning examples	1-14
1.6.1	Planning example for DP slave	1-14
1.6.2	Planning example for DP master	1-24
1.7	PROFIBUS-DP diagnosis	1-38
1.7.1	Status LEDs	1-38
1.7.2	PROFIBUS error messages	1-39
1.7.3	Function blocks	1-39
1.8	Further information	1-40
1.8.1	Standardization	1-40
1.8.2	Important addresses	1-40
1.8.3	Terms, definitions and abbreviations used	1-40
2	Figures	2-1
3	Tables	3-1
4	Index	I

1 The PROFIBUS-DP coupler

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

PROFIBUS-DP is designed for the rapid transfer of process data between central controller modules (such as PLC or PC) and decentralized field modules (such as I/O modules, drives and valves) in the field level. The communication occurs mainly cyclic. For intelligent field modules, additionally acyclic communication functions are required for parameter assignment, diagnosis and alarm handling during the running cyclic data transfer.

During normal operation a central controller (DP master of class 1) cyclically reads the input data of the connected decentralized I/O modules (DP slaves) and sends output data to them. Per slave a maximum of 244 bytes of input and output data can be transferred in one cycle.

Apart from the user data traffic PROFIBUS-DP provides extensive commissioning and diagnosis functions. The present diagnosis messages of all slave modules are summarized in the master. This enables a quick localization of errors.

Using PROFIBUS-DP both mono-master and multi-master systems can be realized. Multi-master systems are built of functional independent subsystems which each consist of one master and a portion of the slaves which are integrated in the entire system. Normal bus masters cannot exchange information with each other.

PROFIBUS-DP distinguishes two types of masters. The class 1 master carries out the cyclic transfer of user data with the slaves and supplies the user data. The class 1 master can be called by a class 2 master using specific functions. These functions are restricted services, for example the interrogation of diagnosis information of the slaves or the master itself. Thus a class 2 master is also considered as a programming and diagnosis device.

PROFIBUS-DP uses the hybrid bus access method. This guarantees on the one hand that complex automation devices used as DP masters obtain the opportunity to handle their communication tasks in defined time intervals. On the other hand it enables the cyclic and real-time related data exchange between the master and peripheral devices (DP slaves). The assigned slave modules on the bus are handled by the master one after the other using the polling operation mode. So, each slave becomes active only after it was requested by the master. This avoids simultaneous access to the bus.

The hybrid access method of the PROFIBUS allows a combined operation of multiple bus masters and even the mixed operation of PROFIBUS-DP and PROFIBUS-FMS in one bus section. This, however, assumes the correct configuration of the bus system and the unique assignment of the slave modules to the masters.

The characteristic properties of a PROFIBUS-DP module are documented in form of an electronic data sheet (modules master data file, GSD file). The modules master data describe the characteristics of a module type completely and clearly in a manufacturer independent format. Using this defined file format strongly simplifies the planning of a PROFIBUS-DP system. Usually the GSD files are provided by the module's manufacturer. In addition, the PROFIBUS user organization (PNO) makes the GSD files of numerous PROFIBUS-DP modules available for a free of charge download via internet in their GSD library. The address of the PROFIBUS user organization (PNO) is: <http://www.profibus.com>.

1.1.2 Features

Modes of operation:

07 KT 97 / 07 KT 98:

- Operation as DP master of class 1 or as DP slave, as desired.
- Automatic setting of the operation mode during configuration.
- Operation mode DP master of class 1:
 - up to 244 bytes of input data and 244 bytes of output data per slave
 - 57344 I/O points max.
- Operation mode DP slave:
 - up to 244 bytes of input or output data each, together max. 368 bytes
 - 2944 I/O points max.
 - automatic baud rate detection

07 SL 97:

- Operation mode DP master of class 1
 - up to 244 bytes of input data and 244 bytes of output data per slave
 - 16384 I/O points max.

Transmission technique:

- RS-485, potential-separated, insulation voltage up to 850 V.
- Twisted pair cable or optical fibre as a medium for the bus.
- Transfer rate from 9.6 kbit/s up to 12 Mbit/s.
- Bus length up to 1200 m at 9.6 kbit/s and up to 100 m at 12 Mbit/s.
- Up to 32 participants (master and slave modules) without repeaters and up to 126 participants on one bus with repeaters.
- 9-pole SUB-D socket for bus connection; assignment according to standard.
- Integrated repeater controller.
- Automatic baud rate detection in operation mode DP slave.

Communication:

- Cyclic user data transfer between DP master and DP slave.
- Acyclic data transfer from master to master.
- Slave configuration check.
- Efficient diagnosis functions, 3 graduated diagnosis messaging levels.
- Synchronization of inputs and/or outputs via control commands.

Protection functions:

- Message transfer with Hamming distance HD = 4.
- Errors in the data transfer are detected by the CRC check and cause the repetition of the telegram.
- Access protection for inputs and outputs of the slaves.
- Incorrect parameter settings are avoided because participants with faulty parameters are not included in the user data operation.
- A failure of a participant is registered in the master and indicated via a common diagnosis.
- Response monitoring for DP slaves. Failure of a transmission line is detected and causes the outputs to be switched off.

Status indication via 4 LEDs:

- READY (yellow): The coupler is ready for operation.
- RUN (green): Status of configuration and communication.
- ERROR (red): Error on PROFIBUS.
- STATUS (yellow): Data exchange.

1.2 Technical data

1.2.1 Technical data of the coupler

Coupler type	PROFIBUS coupler in PC/104 format
Processor	16 bit processor with interrupt controller and DMA controller
Memory expansion	07 KT 97: 8 kB DPR, 512 kB Flash-EPROM, 128 kB RAM 07 SL 97: 2 kB DPR, 512 kB Flash-EPROM, 128 kB RAM
Internal power supply with	+5 V, 650 mA
Dimensions	96 x 90 x 23 mm
CE sign	55011 Class B for Emissions, EN 50082-2 for Immunity

1.2.2 Interface specification

Interface socket	9-pin, SUB-D socket
Transmission standard	EIA RS-485 acc. to EN50170, potential-free
Transmission protocol	PROFIBUS-DP, max. 12 Mbaud
Transmission rate	Baudrate 9,6 kbit/s up to 12000 kbit/s
Status indication	by 4 LEDs
Number of participants (master/slave modules per bus segment)	32 max.
Number of participants via repeaters	126 max.

1.3 Connection and transfer media

1.3.1 Attachment plug for the bus cable

9-pin SUB-D connector

Assignment:

Pin No.	Signal	Meaning
1	Shield	Shielding, protective earth
2	not used	
3	RxD/TxD-P	Receiving / sending line, positive
4	CBTR-P	Control signal for repeater (optional)
5	DGND	Reference potential for data lines and +5V
6	VP	+5 V, supply voltage for bus terminating resistors
7	not used	
8	RxD/TxD-N	Receiving / sending line, negative
9	CNTR-N	Control signal for repeater, negative (optional)

Table 1-1 Pin assignment of the attachment plug for the bus cable

Supplier:

e.g. Erbic® BUS Interface Connector

ERNI Elektroapparate GmbH

Seestraße 9

D-72099 Adelberg

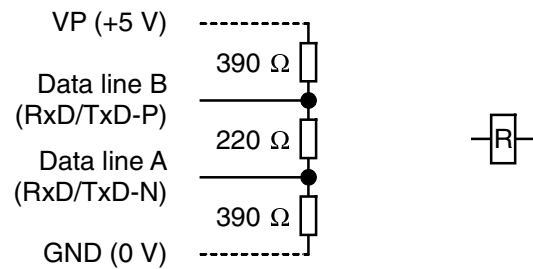
Phone: (+49) 7166 50 176

Fax: (+49) 7166 50 103

Internet: <http://www.erni.com>

1.3.2 Bus terminating resistors

The line ends (of the bus segments) have to be terminated using bus terminating resistors according to the drawing below. The bus terminating resistors are usually placed inside the bus connector.



Configuration of the resistors

Symbol

Figure 1-1 Bus terminating resistors connected to the line ends

1.3.3 Bus cable

Type:	Twisted pair cable (shielded)
Characteristic impedance (cable impedance):	135...165 Ω
Cable capacity (distributed capacitance):	< 30 pF/m
Diameter of line cores (copper):	≥ 0.64 mm
Cross-section of line cores:	≥ 0.34 mm ²
Line resistance per core:	≤ 55 Ω/km
Loop resistance (resistance of 2 cores):	≤ 110 Ω/km

Supplier:

e.g. UNITRONIC® BUS
U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
D-70565 Stuttgart
Phone: (+49) 711 7838 01
Fax: (+49) 711 7838 264
Internet: <http://www.lappkabel.de>

1.3.4 Maximum line lengths (bus segment)

1200 m	at a transfer rate of 9.6 / 19.2 / 93.75 kbit/s
1000 m	at a transfer rate of 187.5 kbit/s
400 m	at a transfer rate of 500 kbit/s
200 m	at a transfer rate of 1500 kbit/s
100 m	at a transfer rate of 3000 / 6000 / 12000 kbit/s

Branch lines are generally permissible for baud rates of up to 1500 kbit/s. But in fact they should be avoided for transmission rates higher than 500 kbit/s.

1.3.5 Repeaters

One bus segment can have up to 32 participants. Using repeaters a system can be expanded to up to 127 participants. Repeaters are also required for longer transfer lines. Please note that a repeater's load to the bus segment is the same as the load of a normal participant. The sum of normal participants and repeaters inside of one bus segment must not exceed 32.

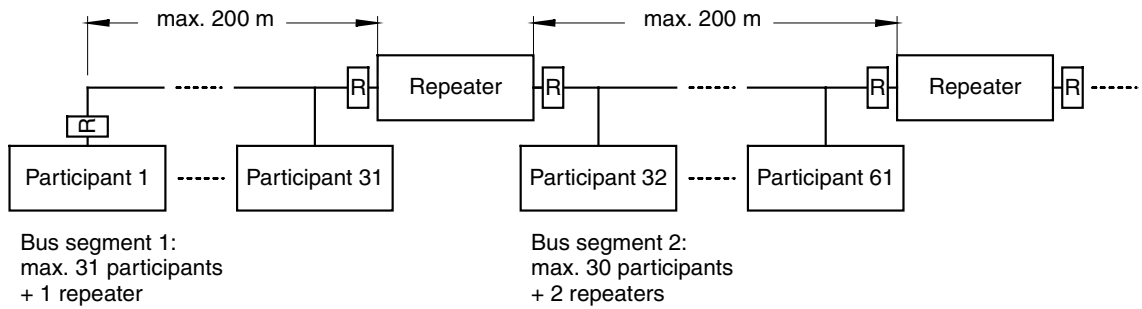


Figure 1-2 Principle example for a PROFIBUS-DP system with repeaters (1500 kbit baud rate)

1.4 Possibilities for networking

The PROFIBUS coupler is connected to the bus via the 9-pin SUB-D socket. For EMC suppression and for protection against dangerous contact voltages, the shield of the bus line has to be connected to protective earth outside the housing.

1.4.1 Single master system

The single master system is the simplest version of a PROFIBUS network. It consists of a class 1 DP master and one or more DP slaves. Up to 31 DP slaves can be connected to the bus without using a repeater. If the number of bus segments is increased by means of repeaters, up to 126 DP slaves can be handled. The line ends of the bus segments have to be terminated using bus terminating resistors.

The DP master of class 1 is able to:

1. Parameterize DP slaves (e.g. timing supervision, bus interchange).
2. Configure DP slaves (e.g. type / number of channels).
3. Read input and output data of the DP slaves.
4. Write output data of the DP slaves.
5. Read diagnosis data of the DP slaves.
6. Send control commands to the DP slaves (e.g. freezing input signals).

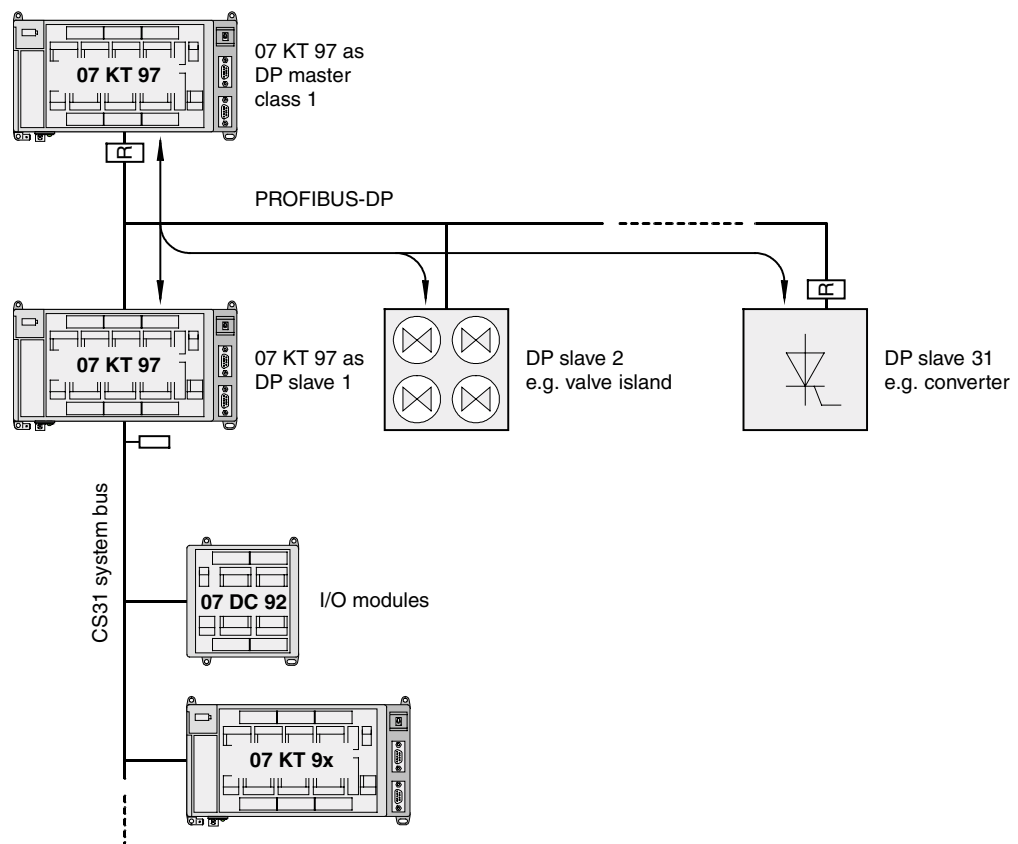


Figure 1-3 Single master system example

1.4.2 Multi-master system

A PROFIBUS network containing several DP masters is called a multi-master system. Up to 32 participants (DP masters and DP slaves) can be operated on one bus segment. Using repeaters the system can be expanded to up to 127 participants. In a multi-master system no data exchange between the DP masters is performed. The entire system is divided into logical subsystems inside of which one DP master communicates with the assigned DP slaves. Each DP slave can be assigned to only one DP master. The master has unlimited access to its assigned slaves while all other masters on the bus can only read the input and output data of these slaves.

All DP masters of class 1 (normal bus master, here 07 KT 97) and class 2 (commissioning device, typically a PC) can read the input and output data of all slaves.

Additionally the DP masters of class 1 and class 2 have the following access possibilities to their assigned DP slaves. They are able to:

- Parameterize DP slaves (e.g. timing supervision, bus interchange).
- Configure DP slaves (e.g. type / number of channels).
- Write output data of the DP slaves.
- Read diagnosis data of the DP slaves.
- Send control commands to the DP slaves (e.g. freezing input signals).

A DP master of class 2 is additionally able to:

- Read and write configuration data of the class 1 DP masters.
- Read configuration data of the DP slaves.
- Read diagnosis data of the class 1 DP masters.
- Read out the diagnosis data of the DP slaves assigned to the respective DP master.

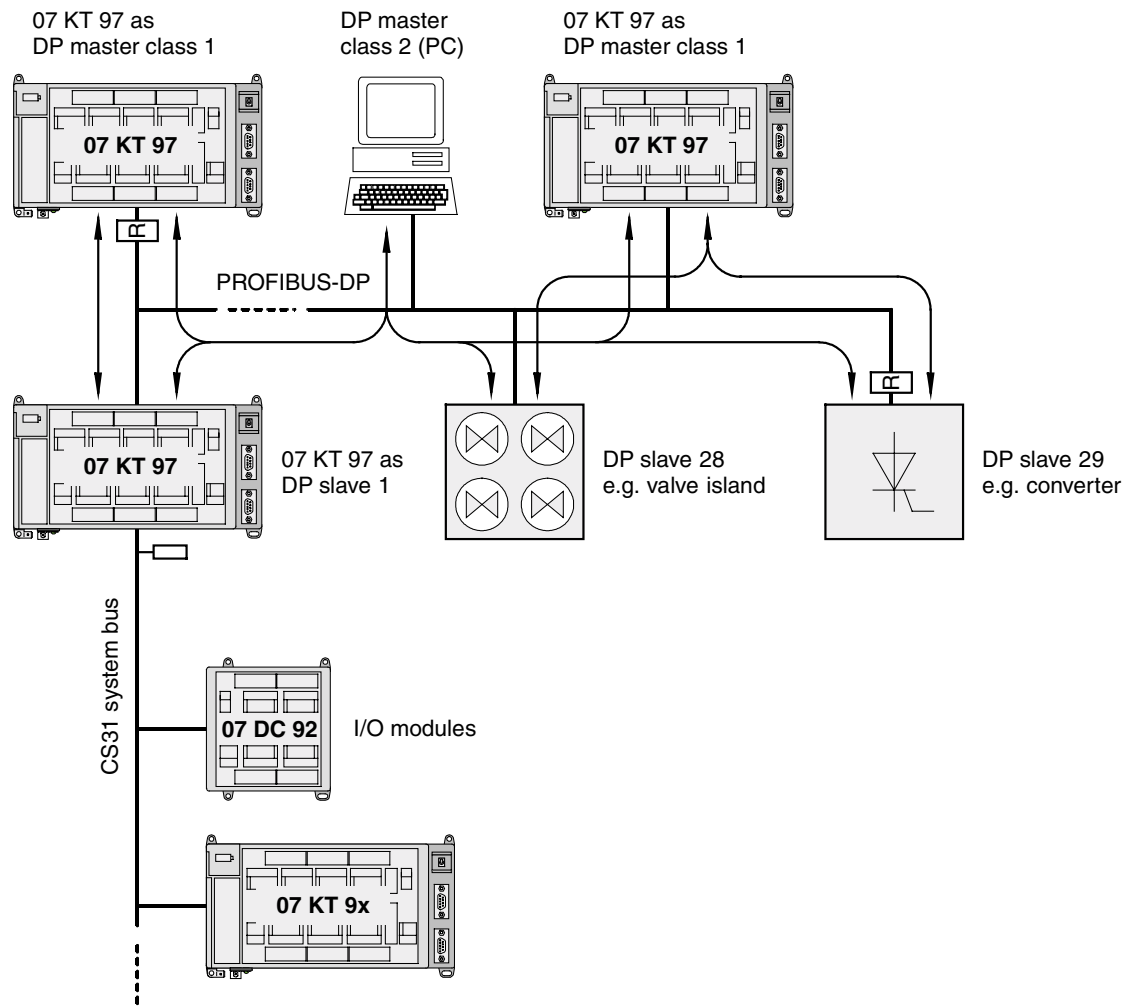


Figure 1-4 Multi-master system example

1.5 PROFIBUS implementation

1.5.1 Configuration

A correct configuration is an assumption for the correct function of the PROFIBUS coupler. There are two possible methods for configuring the PROFIBUS coupler. One method is to use the PROFIBUS configurator of the 907 AC 1131 programming software and the other method is to configure the coupler by means of the field bus coupler configuration software 907 FB 1131. If the PROFIBUS coupler is exclusively used for several projects, it is advisable to use the configurator of the programming software. The usage of the configuration software 907 FB 1131 is advantageous for projects which should be used with varying field bus couplers. For instance, this is the case for series-wound machines where the network connection of the field bus I/Os with the controller has to be performed customer specific, one time using PROFIBUS or the next time using Interbus.

Configuration using 907 AC 1131

When configuring the PROFIBUS coupler using the PROFIBUS configurator of the 907 AC 1131 software, the configuration data are a definite element of a project. In the 907 AC 1131 software they are created selecting *Resources / Controller configuration* (refer to 907 AC 1131 documentation). The coupler configuration data are transferred to the controller together with the actual program. The operating system of the PLC first carries out a validity check and then transfers the configuration data to the coupler. If an error occurs here, it is displayed via the FK error flags and LEDs. In addition, the configuration data can be saved together with the user program in the Flash memory of the control. Configuration data which are saved in the Flash memory are automatically loaded the next time the power is switched on.

The PROFIBUS coupler selects its operation mode according to the configuration data and signalizes that it is ready for operation. The coupler's readiness for operation is indicated by cyclic flashing of the LED RDY which is located beside the bus connection.

Configuration using 907 FB 1131

Alternatively the PROFIBUS coupler can also be configured using the field bus configurator 907 FB 1131 (refer to 907 FB 1131 documentation). Configuration data created using 907 FB 1131 are not assigned directly to a project. Thus, the resulting file has to be downloaded separately to the controller (additionally to a 907 AC 1131 project) where it is stored in the Flash memory. The PROFIBUS coupler selects its operation mode according to the configuration data and signalizes that it is ready for operation. The coupler's readiness for operation is indicated by cyclic flashing of the LED RDY which is located beside the bus connection. If the user defined project is written to SMC, also the configuration data are automatically saved.



Caution:

Configuration using the field bus configurator 907 FB 1131 is possible from the PLC run time system version V4.16 onwards.

1.5.2 Running operation

The PROFIBUS-DP protocol is automatically handled by the coupler and the operating system of the controller. The coupler is only active on the bus if it was correctly initialized before and if the user program runs. No connection elements are required for the cyclic exchange of process data via PROFIBUS-DP. Special PROFIBUS-DP functions can be realized using the function blocks of the corresponding PROFIBUS library.

1.5.3 Operation mode DP slave

In the DP slave operation mode the coupler only responds to diagnosis requests of the DP master if the user program is not active. By setting the `Station_not_ready` bit in the response telegram, the coupler signalizes to the requesting master that it is not ready for data exchange.

After the user program is started the coupler signalizes that it is ready on the bus and consequently it is added to the cyclic telegram traffic by the DP master. When the master could take it successfully into operation, the LED RDY indicates this by lighting continuously. The coupler stores the data received from the master in the operand areas defined in the configuration and sends the configured process data to the master. If the user program is stopped, the coupler logs off from the bus.

The operation mode DP slave is completely integrated to the operating system of the controller. The transmit or receive data in the corresponding operand areas can be directly accessed. The access can be performed either via operands or symbolically. Function blocks are not required.

The function block library contains a block which can be used to poll various status information of the coupler. If necessary, this block can be inserted additionally.

1.5.4 Operation mode DP master

In the DP master operation mode the coupler starts the communication via PROFIBUS-DP after the user program is started and attempts to initialize the planned slaves. After a slave is initialized successfully it is added to the cyclic process data exchange. The LED RDY lights up continuously after at least one slave was successfully taken into operation. If the user program is stopped, the coupler shuts down the PROFIBUS system in a controlled manner.

The operation mode DP master is completely integrated to the operating system of the controller. The transmit or receive data of the slaves in the corresponding operand areas can be directly accessed. The access can be performed either via operands or symbolically. Function blocks are not required.

The function block library contains various blocks which can be used e.g. to poll status information of the coupler or to execute specific acyclic PROFIBUS-DP functions. If necessary, these blocks can be inserted additionally.

1.5.5 Error diagnosis

PROFIBUS-DP communication errors are generally indicated by the red LED ERR beside the bus connector. A malfunction of the PROFIBUS driver or the coupler itself is indicated via the FK error flags and the corresponding LEDs. Furthermore the PROFIBUS library provides different function blocks which allow a detailed error diagnosis. In both operating modes DP master and DP slave, the state of the coupler itself can be read out via a corresponding connection element. Additionally, in the DP master operation mode the detailed PROFIBUS diagnosis of an individual slave and a system diagnosis overview can be retrieved.

1.5.6 Function blocks

Version of 907 AC 1131	PLC runtime system	Libraries	Remarks
V4.0	V4.0x onwards	PROFI_40.LIB CIF104.LIB	
V4.1	V4.1x onwards	PROFIBUS_S90_V41.LIB Coupler_S90_V41.LIB	Coupler_S90_V41.LIB cannot be used with PLC runtime system V4.0x
V4.2	V4.1x onwards	PROFIBUS_S90_V41.LIB PROFIBUS_S90_V42.LIB Coupler_S90_V41.LIB	Bug removal in PROFIBUS_S90_V41.LIB
V4.3	V4.1x onwards	PROFIBUS_Master_S90_V43.LIB PROFIBUS_Slave_S90_V43.LIB Coupler_S90_V41.LIB	Divided into master library and slave library due to new function blocks

Table 1-2 Overview of PROFIBUS libraries

Libraries for 907 AC 1131 V4.0 ... V4.2:

PROFI_40.LIB and CIF104.LIB or

PROFIBUS_S90_V41(2).LIB and Coupler_S90_V41.LIB

Group	Function block	Function
General		
	PROFI_INFO	Reading of coupler information
DP master		
	DPM_STAT	Reading the coupler status
	DPM_SLVDIAG	Reading the detailed PROFIBUS diagnosis of a slave
	DPM_SYSDIAG	Reading the system diagnosis
DP slave		
	DPS_STAT	Reading the coupler status

Table 1-3 Function blocks contained in the library PROFIBUS_S90_V41.LIB

Coupler_S90_V41.LIB

Contains various internal functions which are used by the corresponding field bus coupler libraries.

Libraries for 907 AC 1131 V4.3 onwards:

PROFIBUS_Master_S90_V43.LIB and Coupler_S90_V41.LIB

Group	Function block	Function
General		
	PROFI_INFO	Reading of coupler information
Status / Diagnosis		
	DPM_STAT	Reading the coupler status
	DPM_SLVDIAG	Reading the detailed PROFIBUS diagnosis of a slave
	DPM_SYSDIAG	Reading the system diagnosis
Control system		
	DPM_CTRL	Sending control commands to slaves
Acyclic reading		
	DPM_READ_INPUT	Reading input data of slaves which are not assigned to the master
	DPM_READ_OUTPUT	Reading output data of slaves which are not assigned to the master

Table 1-4 Function blocks contained in the library PROFIBUS_Master_S90_V43.LIB

PROFIBUS_Slave_S90_V43.LIB and Coupler_S90_V41.LIB

Group	Function block	Function
Status / Diagnosis		
	DPS_STAT	Reading the coupler status

Table 1-5 Function blocks contained in the library PROFIBUS_Slave_S90_V43.LIB

Coupler_S90_V41.LIB

Contains various internal functions which are used by the corresponding field bus coupler libraries.

1.6 Planning examples

1.6.1 Planning example for DP slave

This planning example first describes the configuration of a controller as a DP slave using the PROFIBUS configurator of the software 907 AC 1131. Then the alternative configuration procedure using the configuration software 907 FB 1131 is shown. Conclusively the corresponding PLC program is created.

Below a planning example for the controller 07 KT 97 R120 used as DP slave is shown. The PLC receives cyclically one word from the master and shall output the value at the analog output AW06,00 / %QW1006.0. The values read at the digital inputs E62,00...E62,07 / %IX62.0...%IX62.7 are to be reported to the master. The bus address no. 2 of the slave is predetermined.

Configuration using 907 AC 1131

Start the 907 AC 1131 software and create a new project for a controller 07 KT 97. Save the empty project as *PROFIBUS-Slave-Example*. Before the actual user program can be built, the behavior of the coupler on the bus must be determined. For this purpose the hardware configuration part of 907 AC 1131 must be called by selecting *Resources / Controller configuration*.

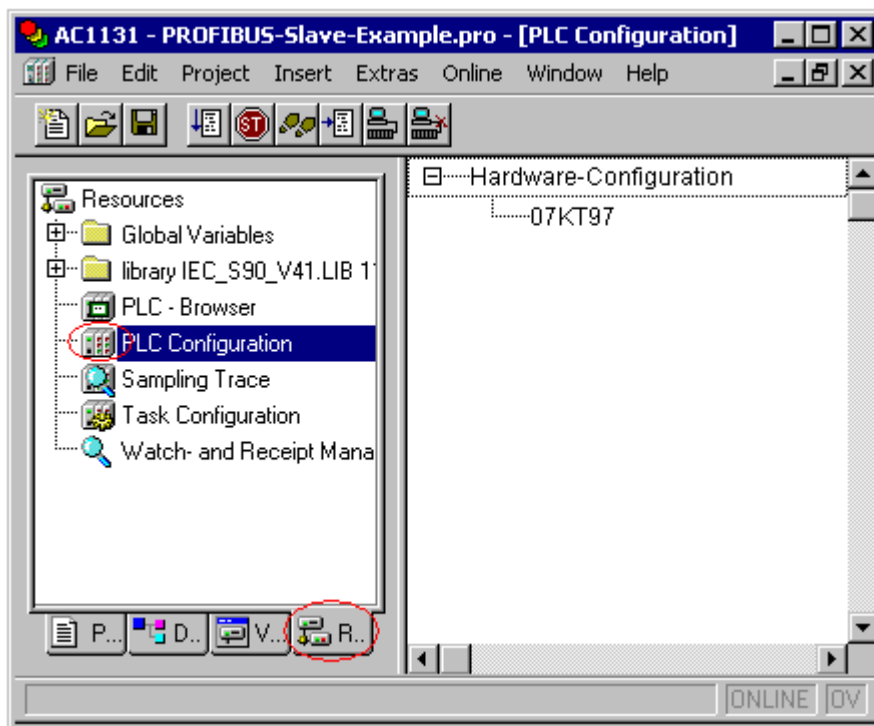


Figure 1-5 Controller configuration in 907 AC 1131

Select *Hardware configuration* and press the right mouse button. Selecting *Append Subelement / DP Slave* from the context menu opens a window that displays all DP slaves whose GSD files are available. In this dialog, select the module name 07 KT 97-DPS. The module number to be selected depends on the slot where the PROFIBUS coupler is installed. This is slot 1 for the 07 KT 97 R0120 or slot 2 for the 07 KT 97 R162. Thus, in this example the module number 1 has to be entered. After you have confirmed the selection with *OK* the coupler properties for this project must be determined.

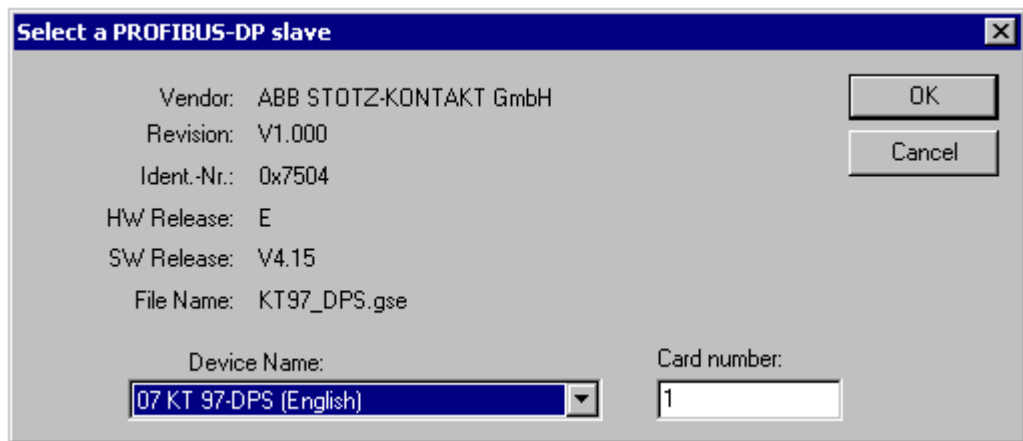


Figure 1-6 Hardware configuration / DP slave

To specify the coupler properties, select the previously inserted entry *07 KT 97-DPS* in the controller configuration tree and press the right mouse button. Select *Properties* from the appearing context menu.

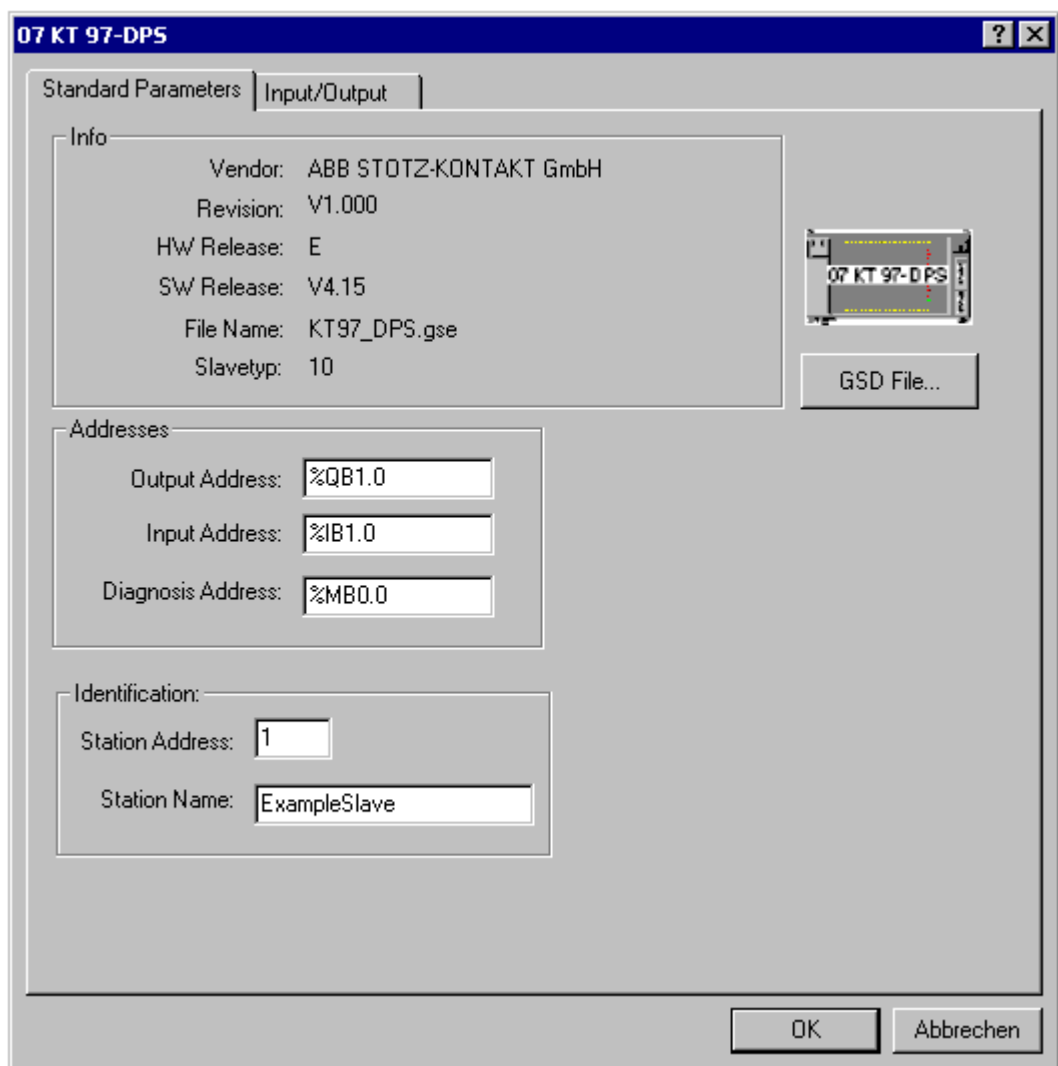


Figure 1-7 DP slave properties

As basic parameters first the start addresses of the input and output data in the operand area have to be preset. The input field for the diagnosis address is without function. Because a word should be received from the master, the input address %IW1.0 is selected in this example. The

values of the digital inputs %IX062.0...%IX062.7 (E62,00...E62,07) are packed in one byte for the transfer to the master. Hence, %QB1.0 is selected for the output address. According to the default setting the value 2 has to be entered for the station address. Now a station name can be assigned optionally, e.g. ExampleSlave. It is not necessary to enter the transfer rate because this is preset by the DP master and automatically detected by the coupler.

Now the I/O configuration has to be determined. For this purpose, open the *Input/Output* tab of the currently displayed dialog. This tab shows two list areas. On the left-hand side all modules are listed which are available as DP slaves for the 07 KT 97. On the right-hand side the current configuration is displayed. For the I/O configuration it has to be observed that the PLC is considered as a simple I/O device. **The designation of the I/O modules is always performed from the process view.** "1 word output" and "1 x 8 bit input" must be selected for the I/O configuration because the controller receives one word via PROFIBUS-DP which it has to put out to the process and transfers one byte to the master which it reads from the process. For this, highlight the corresponding module in the list and apply your selection with *Select>>*. Conclusively confirm your configuration with *OK*. The coupler configuration is now completed. Furthermore symbolic variable designators for the I/O data can be assigned optionally. Thus, the following hardware configuration results:

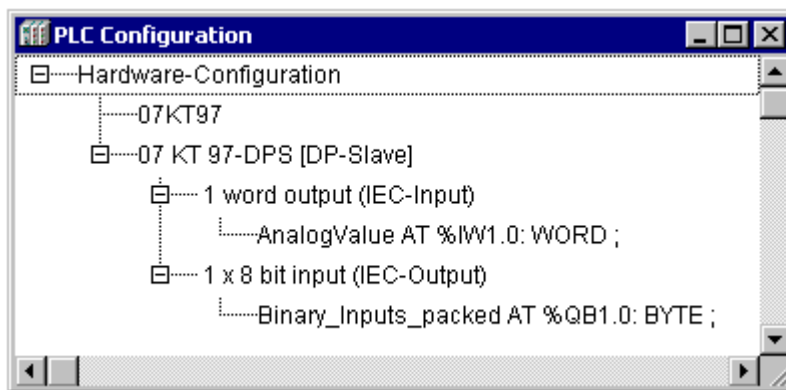



Figure 1-8 Hardware configuration example 07 KT 97 as DP slave

Configuration using 907 FB 1131

Alternatively to the procedure described above the coupler can also be configured using the field bus coupler configuration software 907 FB 1131. For that purpose start the configuration software 907 FB 1131. In the *File* menu select *New*. A window appears listing all available field bus systems. Select *PROFIBUS* and confirm the window with *OK*. The main window now displays an empty bus strand. Save the file under any name to any directory by selecting *File – Save as...* (e.g. *PROFIBUS-Slave-Example*). The extension *.pb* (**P**ROFIBUS) is automatically attached to the file name.

When configuring a coupler using 907 FB 1131 it is not sufficient to describe only the local coupler. In order to allow an optimal design of the timing behavior, the program requires a description of the entire network where the coupler is integrated. Hence, a DP master has to be inserted first to which the 07 KT 97 with PROFIBUS coupler can be assigned as a slave in the next step. In this example a 07 KT 97 shall be used as DP master. If you want to use another DP master, you first have to import the corresponding GSD file (refer to 907 FB 1131 documentation).

Now insert the 07 KT 97 as DP master. For that purpose click on the button *Insert master* . Select *07 KT 97-DPM* in the *Available masters* list and click on the *Add >>* button. After this, the 07 KT 97 is inserted as DP master into the *Selected masters* list. Now you have to specify the desired station address (1 in this example), define a unique description for the device (e.g. *ExampleMaster*) and confirm with *OK*.

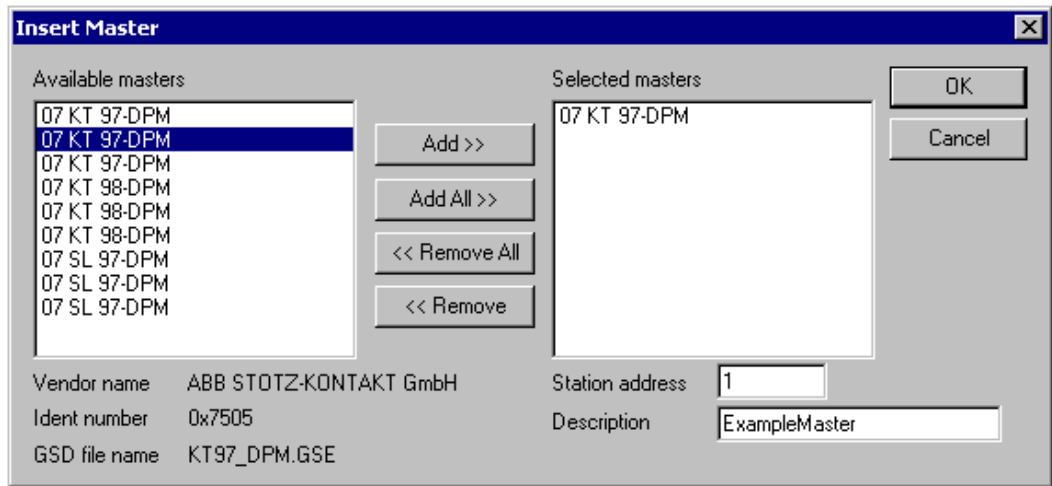



Figure 1-9 Inserting a master

Now insert the 07 KT 97 as DP slave in the same way. Select the button *Insert slave*  and then click into a main window area below the previously inserted master. Select *07 KT 97-DPS* in the *Available devices* list and click on the *Add >>* button. If the desired entry is not visible, set the *Slave filter* for *Vendor* and *Slave Type* to *All*. If, on the other side, the list is confusing because it contains too many entries you can reduce the number of displayed entries by pre-selecting the desired vendor (shown: ABB STOTZ-KONTAKT GmbH) and/or slave type (shown: PLC). After adding, the 07 KT 97-DPS is inserted as DP slave into the *Selected slaves* list. Now specify the desired station address and define a unique device description. For the example given here, enter e.g. station address 2 and the description ExampleSlave according to the definitions made before. Confirm with *OK*.

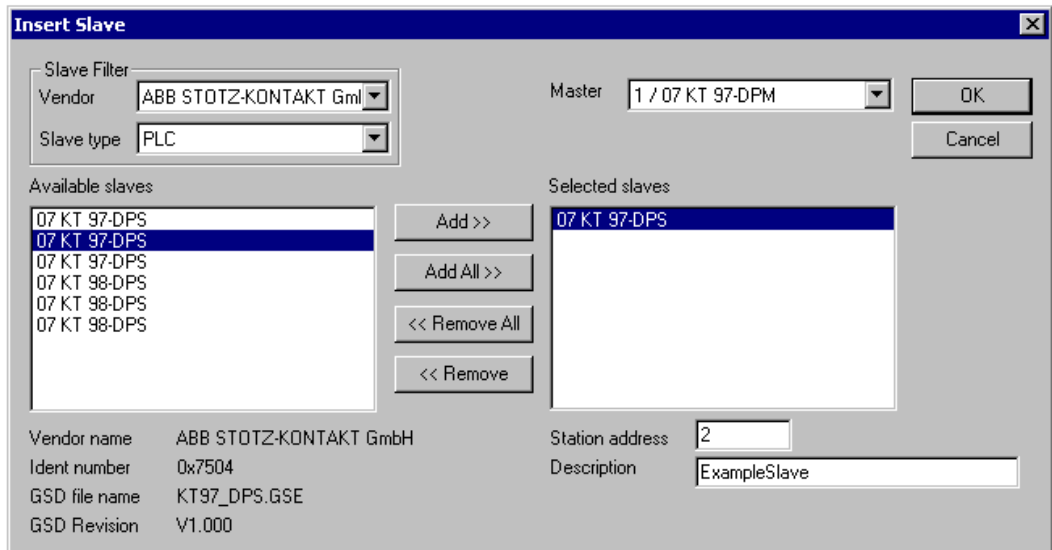


Figure 1-10 Inserting a slave

Thus, the following network configuration results:

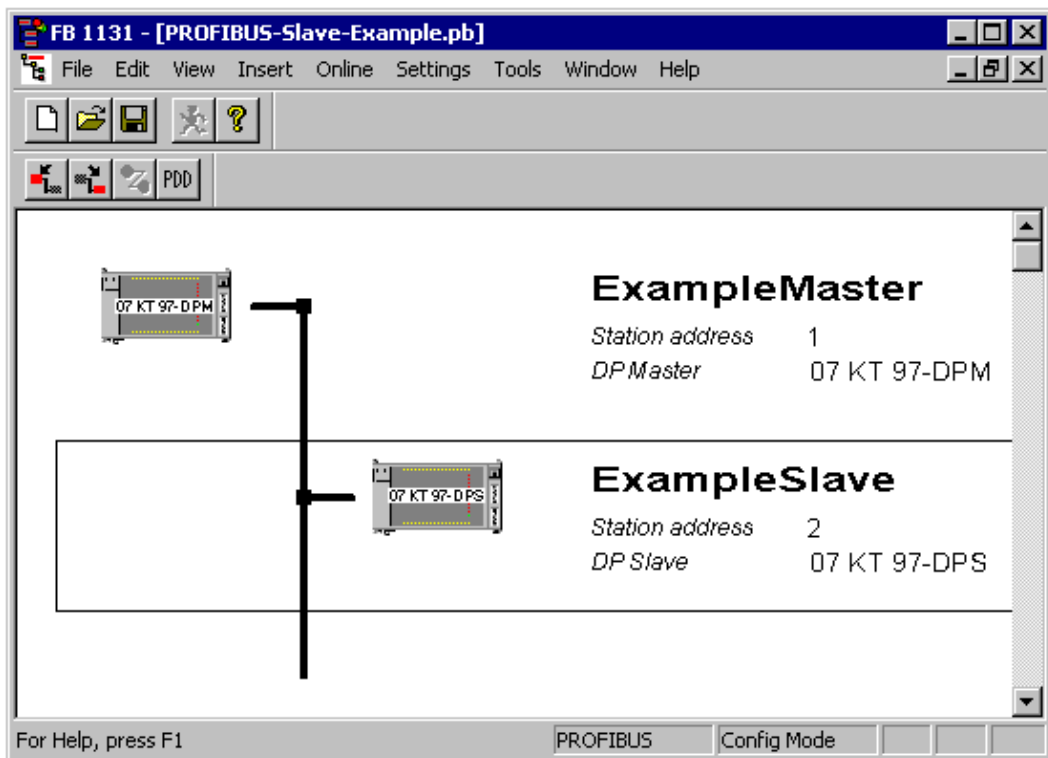


Figure 1-11 Bus overview

In the previous steps the subscribers connected to the bus were described. Now, the communication relationships between these subscribers have to be set (I/O configuration).

For that purpose select the 07 KT 97 which has to be configured as a slave (refer to the figure shown above). In the marked area press the right mouse button and select *Slave configuration* in the appearing context menu. The following window appears.

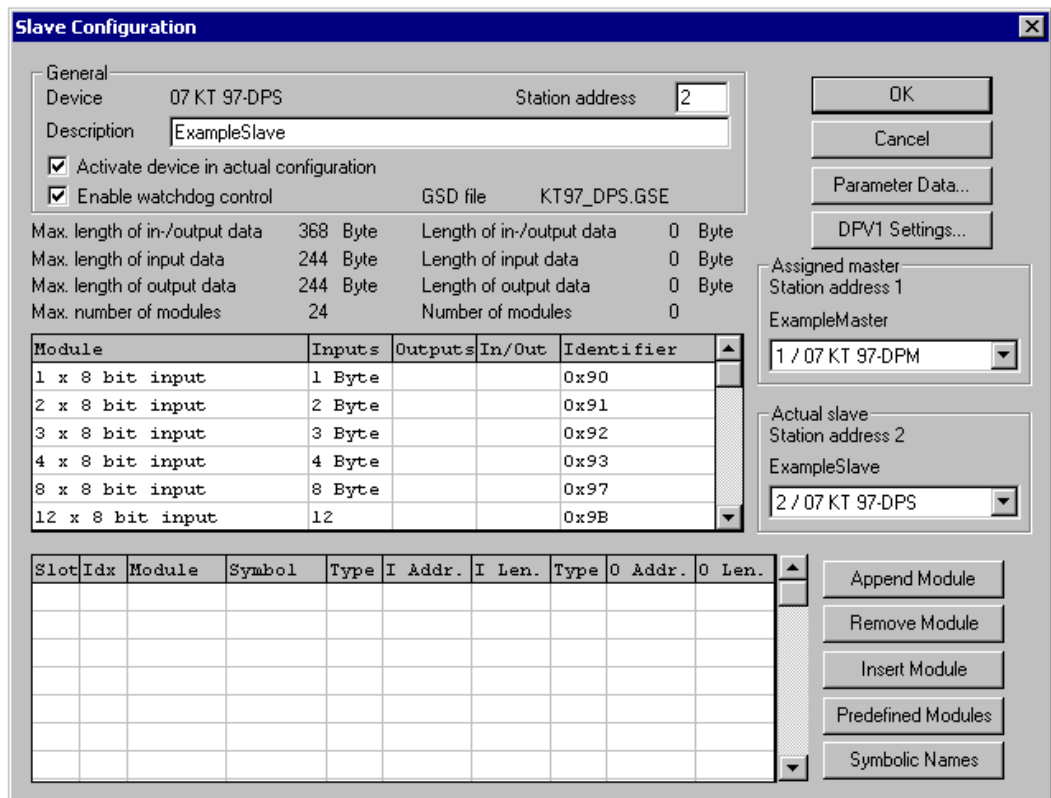


Figure 1-12 Configuring the coupler as DP slave

In this view the communication relationships between the selected slave and the master are defined. In the example given here, the PLC shall receive one word from the master and send 8 bits to the master (see descriptions above). The list of available modules displays all I/O module types which are supported as DP slave by the 07 KT 97. Note, that the designation of the transmission direction (inputs/outputs) always reflects the process view. This means that inputs are data which a remote slave receives from the process and then sends to the master or which the master receives from a slave. Outputs are data which a master sends to a slave or which a slave receives from a master and then outputs to the process on-site. According to the definition that the 07 KT 97 configured as DP slave receives one word from the DP master which should be output as AW6,0 to the process, the entry *1 word output* must be highlighted in the list and selected by clicking on *Add*. The selected data type is then displayed in the lower list of configured I/O modules. In the opposite direction the PLC shall read the digital inputs E62,00...E62,07 from the process and transfer these values in the form of one byte to the master (remark: for PROFIBUS, no bit modules can be defined). Thus, highlight the entry *1 x 8 bit input* in the upper list and apply your selection with *Add*. The I/O configuration of the slave 07 KT 97 is now completed and can be confirmed with *OK*. In our example the further possible settings are not described in detail. For their meanings please refer to the field bus configurator 907 FB 1131 documentation.

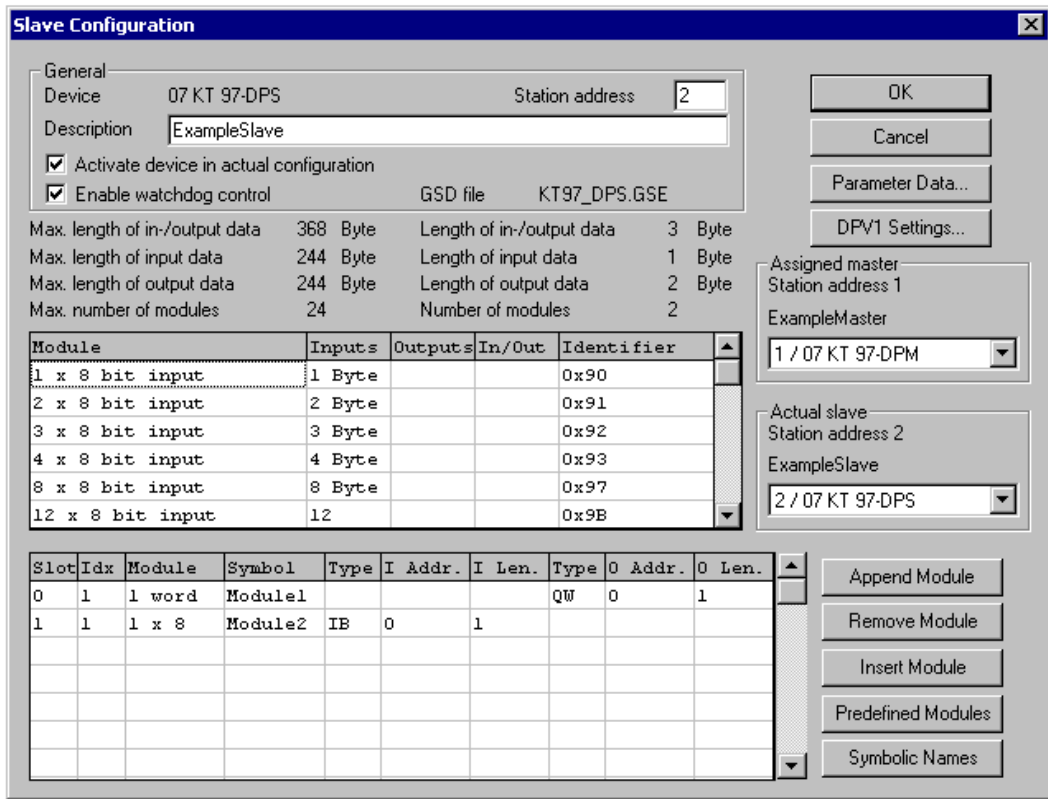


Figure 1-13 Completed configuration of the coupler as DP slave

Finally the data transmission rate has to be set. For this, highlight the master in the main window and select *Settings - Bus Parameters*. A window appears displaying a selection of transmission rates which are supported by all subscribers in the network. Select the desired baud rate and confirm with **OK**. The configuration is now completed and should be saved once more.

In order to download the configuration data to the controller, highlight the 07 KT 97 configured as slave and then select *Settings - Device Assignment*. The dialog for configuring the gateway appears (refer to documentation System technology of the basic units S90).

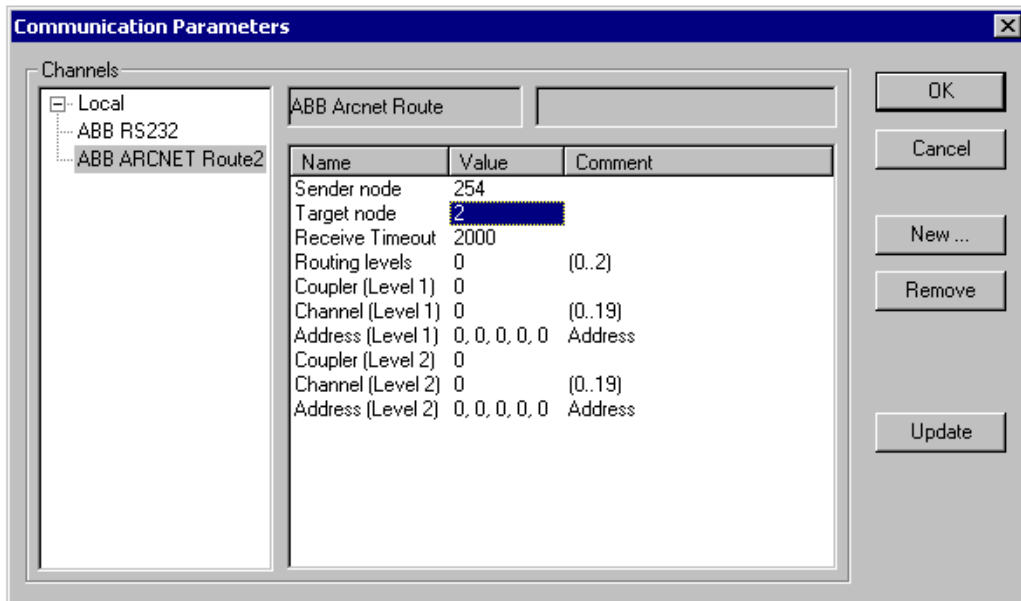


Figure 1-14 Gateway configuration

By selecting *Configure Gateway* you can either open a channel which has already been defined or you can specify a new channel. For transmitting the configuration data the following drivers are available:

- ABB Arcnet Route
- ABB SL97
- ABB RS232 Route

Set the parameters according to the controller to which the data are to be downloaded. Doing this, it is important to specify the correct slot number of the desired coupler. For the controller 07 KT 97 R 120 used in this example, enter slot 1. Select *Connect* to open the set communication channel. If the connection has been established successfully, various information concerning the detected coupler are displayed in the *Device information* area and the field *Errors* contains the value 0. If no communication was possible, the field *Errors* contains a numerical code specifying the occurred error (refer to the field bus configurator 907 FB 1131 documentation). After the connection has been established successfully, you have to confirm the selection of the coupler by *marking it with a cross* and then clicking on *OK*. Now the previously selected channel is assigned uniquely to the opened configuration file. Select *Online - Download* and confirm the following security dialog with *OK*. A window appears displaying the download progress. The window is closed automatically after the download is completed.

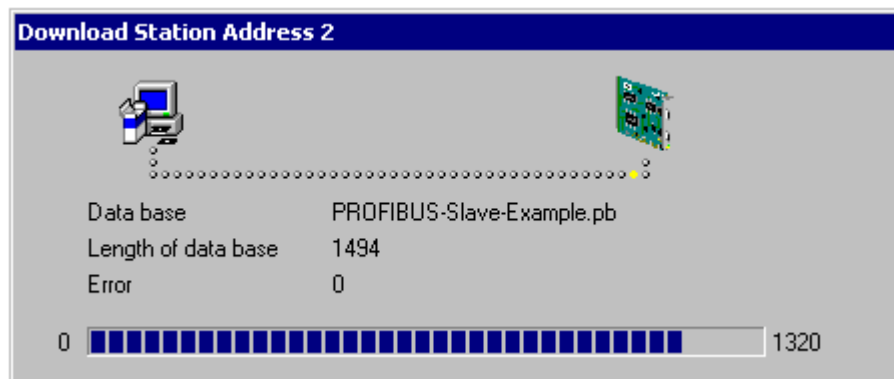


Figure 1-15 Downloading the configuration to the controller

If the configuration data are sent to the coupler without any error, the successful download is additionally indicated by the coupler's green RUN LED which flashes cyclically.

Importing the configuration to 907 AC 1131

Prior to the creation of the actual 907 AC 1131 project, it has to be determined in which areas the PROFIBUS I/O data are stored by the coupler (refer to field bus configurator 907 FB 1131 documentation / section Importing the 907 FB 1131 configuration to 907 AC 1131).

For each internal field bus coupler (except Arcnet) an own operand area is reserved in the controller (refer to documentation System technology of the basic units / Operands) where the corresponding coupler stores the data received via the bus (%I area) or where the data which the coupler has to send via the bus (%Q area) must be stored in the user defined program. Apart from the transmission direction (%I, %Q) the areas are identified by the slot number of the coupler. For the controller 07 KT 97 R 120 configured as DP slave used in this example the PROFIBUS coupler is located in slot 1. Thus, the coupler uses the input data range %IX1.? / %IB1.? / %IW1.? and the output data range %QX1.? / %QB1.? / %QW1.?. Within these ranges the actually transmitted data have to be specified. According to the related I/O configuration the coupler uses the operands separated for each transmission direction, each beginning with an offset of 0. Accordingly, a variable of the data type WORD has to be declared in the 907 AC 1131 project which is located within the input operand range of the internal coupler in slot 1 (%IW1.?). The variable has the offset address 0 WORD (%IW1.0) within this range. Analogous, a variable of the data type BYTE which is located within the output operand range of the internal coupler in slot 1 (%QB1.?) has to be declared for the value sent to the master. The variable has the offset address 0 BYTE (%QB1.0) within this range.

For that purpose start 907 AC 1131. Create a new project for a controller of the type 07 KT 97 and save it as *PROFIBUS-Slave-Example*. Select the *Resources* view and create a *New Object* named *FieldbusIOs* in the *Global Variables*. Define 2 variables with any name which meet the requirements given above concerning the data types and operand addresses.

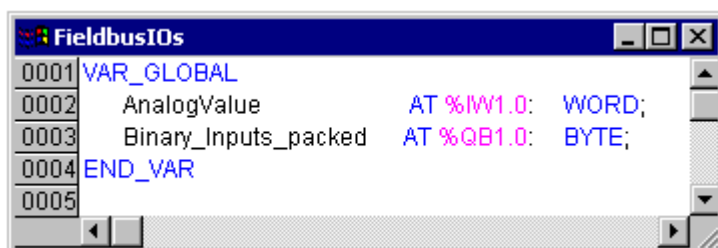


Figure 1-16 Variables declaration

Having done these entries, the PROFIBUS I/Os can be used within the 907 AC 1131 project.

Developing the PLC program using 907 AC 1131

Now the user program must be developed which implements the required behavior of the controller. At the beginning, the inputs and outputs of the controller can not be called using their symbolic names but only as IEC operators. So the IEC operators %IX62.0 - %IX62.7 correspond to the digital inputs E62,00...E62,07 and %QW1006.0 corresponds to the analog output AW6,0. Alternatively the digital inputs E62,00...E62,07 can also be called packed in a byte (E62,08...E62,15 = %IB125) simultaneously via %IB124. If you are not familiar with this representation, symbolic variables can be assigned to the inputs and outputs by declaring global variables using *Resources / Global variables*. These global variables can also be accessed anywhere in the program. The declaration could be as follows:

```

0001 VAR_GLOBAL
0002 (* Declaration Analog Output AW6,0 *)
0003   AW06_00 AT %QW1006.0 : INT;
0004
0005 (* Declaration Biary Inputs E62,00..E62,07 - Bits *)
0006   E62_00 AT %IX62.0 : BOOL;
0007   E62_01 AT %IX62.1 : BOOL;
0008   E62_02 AT %IX62.2 : BOOL;
0009   E62_03 AT %IX62.3 : BOOL;
0010   E62_04 AT %IX62.4 : BOOL;
0011   E62_05 AT %IX62.5 : BOOL;
0012   E62_06 AT %IX62.6 : BOOL;
0013   E62_07 AT %IX62.7 : BOOL;
0014
0015 (* Declaration Binary Inputs E62,00..E62,07 - Byte *)
0016   E62_00_07 AT %IB124.0 : BYTE;
0017 END_VAR
0018

```

Figure 1-17 Example for the declaration of global variables

In order to output the analog value received via PROFIBUS-DP to AW6,0 the following assignment is required in the program:

```
AnalogValue----AW6_00
```

Alternatively and without a previous variable declaration also the following assignment can be used:

```
%IW1.0----%QW1006.0
```

For transferring the digital inputs E62,00...E62,07 to the master also different possibilities exist. If the inputs are called as a group via E62_00_07 only the following assignment is required to transfer the values to the master:

```
E62_00_07----Binary_Inputs_packed
```

The direct assignment of an individual input to a PROFIBUS output operand is also possible:

```
%IX62.0----%QX1.0.0
```

or

```
E62_00----%QX1.0.0
```

or

```

VAR
PROFIBUS_BIT0 AT %QX1.0.0: BOOL;
END_VAR

```

```
E62_00----PROFIBUS_BIT0;
```

After this, the creation of the PLC program is completed. Additionally you can insert the function block DPS_STAT to the project in order to read out the coupler status. Now the project can be downloaded to the controller and tested.

1.6.2 Planning example for DP master

This planning example first describes the configuration of a controller as a DP master using the PROFIBUS configurator of the software 907 AC 1131. Then the alternative configuration procedure using the configuration software 907 FB 1131 is shown. Conclusively the corresponding PLC program is created.

Below, a planning example for the 07 KT 97 R162 used as DP master is shown. Beside the 07 KT 97 which is used as the DP master, one valve island and the controller mentioned in the example above are present on the bus. So the resulting configuration is as follows:

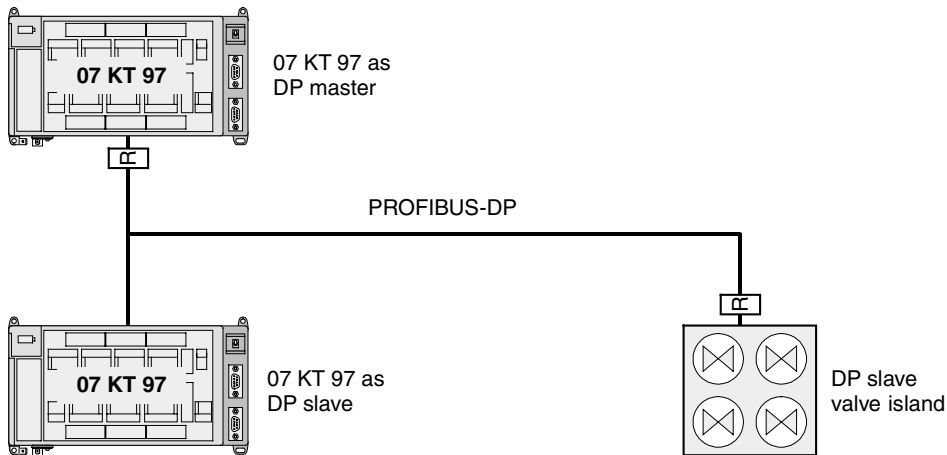


Figure 1-18 PROFIBUS-DP example configuration

The DP master controller sends the difference resulting from the values which are measured at the local analog inputs EW06,00 and EW06,01 to the 07 KT 97 used as DP slave and receives eight binary values packed in one byte from the DP slave. The DP master PLC sends the received binary values as control signals to the valve island (e.g. FESTO type 02 FB 13, 4 valves). The following bus addresses are agreed: 1 for the 07 KT 97 used as DP master, 2 for the 07 KT 97 used as DP slave, 3 for the valve island.

Prior to taking the master PLC into operation it makes sense to configure the slaves first. Proceed as shown in the example above to configure the 07 KT 97 as DP slave. Refer to the manufacturer's documentation for the configuration procedure of the valve island. Such devices normally have DIP switches where the bus address can be set. The I/O configuration is usually fixed and the transfer rate is automatically detected.

Configuration using 907 AC 1131

After the DP slaves have been configured, the configuration of the master can be performed. Start the 907 AC 1131 software. Create a new project for a controller of the type 07 KT 97 and save it as *PROFIBUS-Master-Example*. By selecting *Resources | Controller configuration* the hardware configuration part of the 907 AC 1131 is opened.

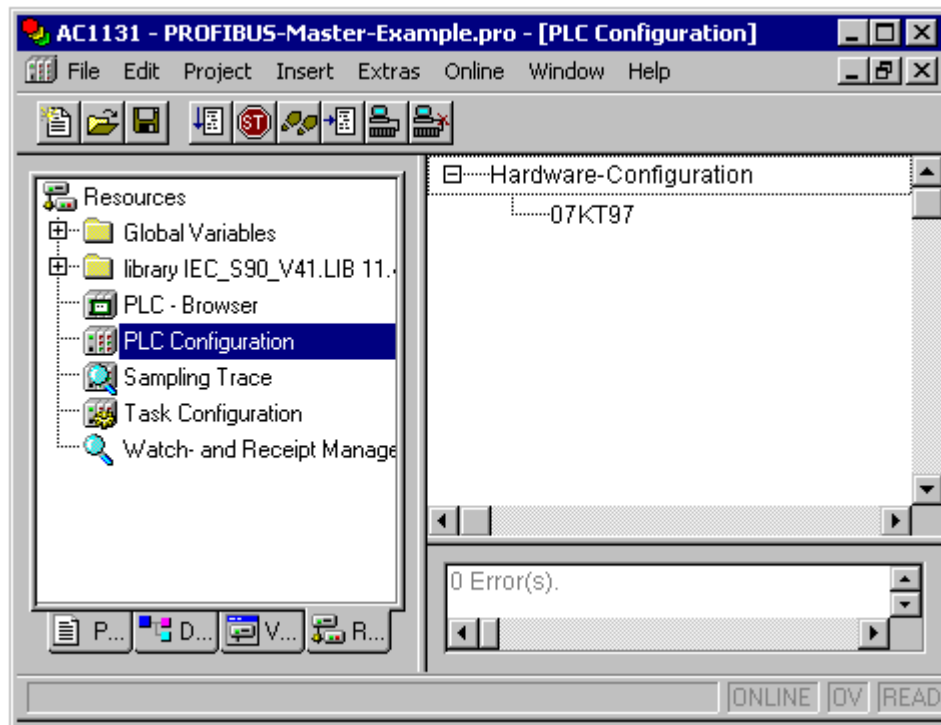


Figure 1-19 Controller configuration

Select *Hardware configuration* and press the right mouse button. Selecting *Append Subelement / DP Master* from the context menu opens a window that displays all DP masters whose GSD files are available. In this dialog, select the module name 07 KT 97-DPM. The module number to be selected depends on the slot where the PROFIBUS coupler is installed. This is slot 1 for the 07 KT 97 R0120 or slot 2 for the 07 KT 97 R162. In this example the 07 KT 97 R162 shall be used, so the module number 2 must be entered. After you have confirmed the selection with *OK* the coupler properties for this project have to be determined.

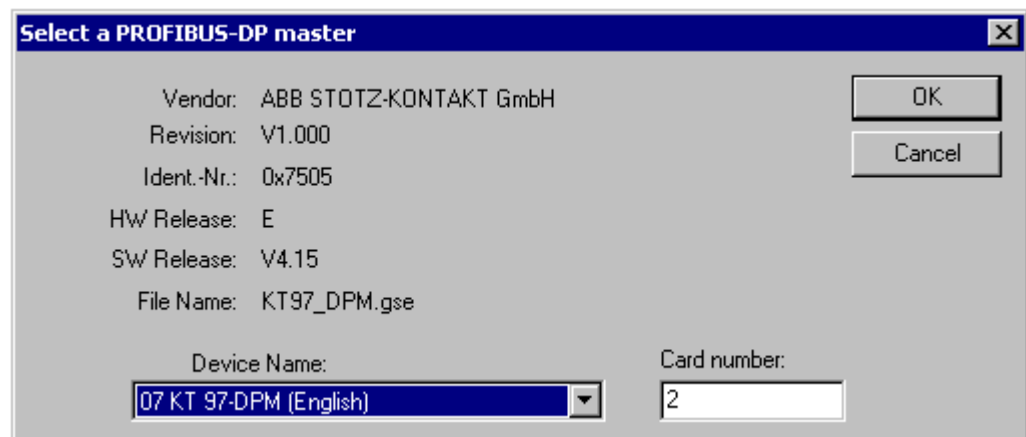


Figure 1-20 Inserting the coupler as a master

To specify the coupler properties, select the previously inserted entry *07 KT 97-DPM* in the controller configuration tree and press the right mouse button. Select *Properties* from the appearing context menu.

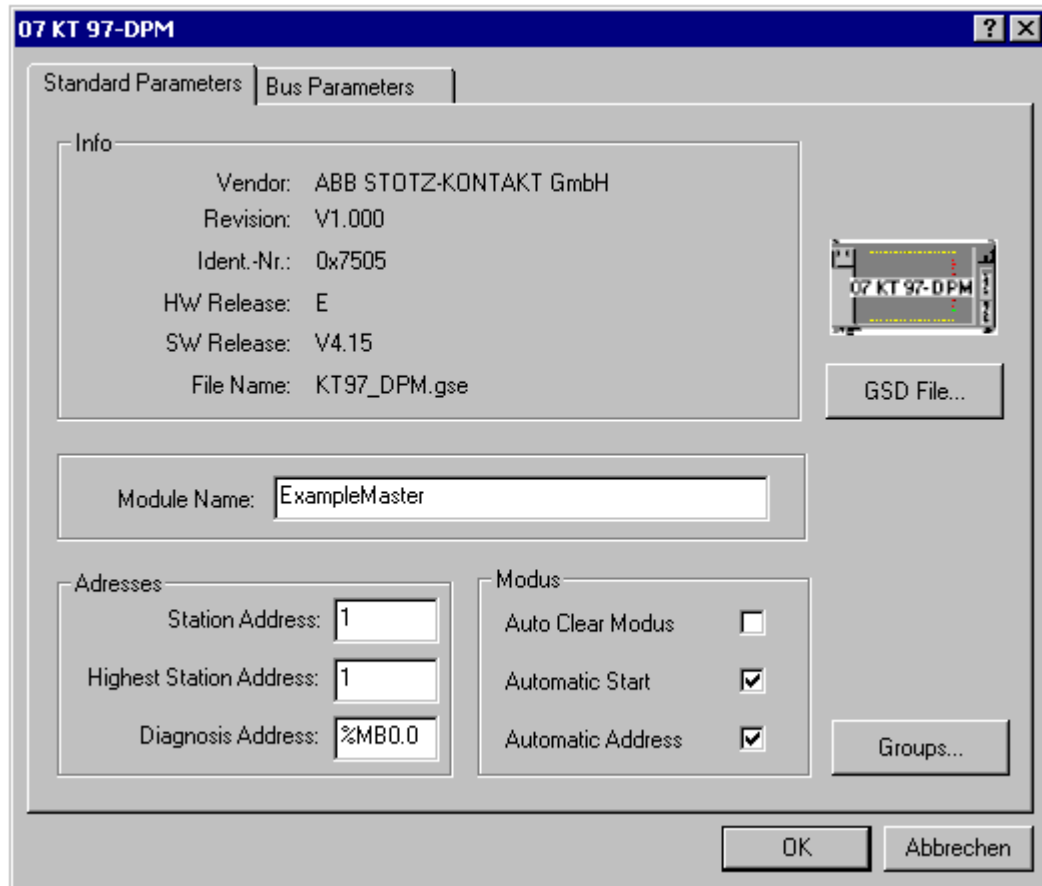


Figure 1-21 Coupler properties when used as master

As basic parameters an individual name and the bus address can be assigned. The highest station address is automatically calculated. Using "Automatic addresses" you can choose whether the IEC operand addresses shall be generated automatically or entered by hand. It is recommended that the address administration is performed automatically by the configurator. In this application example no further settings are required. Refer to chapter Controller configuration in the 907 AC 1131 documentation for further information. Therefore, in this example only the value 1 must be entered for the station address and if necessary an individual name for the master must be assigned (e.g. ExampleMaster). Now open the *Bus Parameters* tab of the currently displayed dialog. This dialog displays a combo box for setting the baud rate and a list view showing the characteristics of the bus system which is currently configured. The baud rate should be set manually to the highest possible value supported by all subscribers on the bus. The selected baud rate must also be manually set for all modules in the system which do not have an automatic baud rate detection. The calculation of the bus parameters should be left to the configurator (*Optimize* checked). Incorrect manual inputs can lead to an undefined system behavior. The settings of the coupler properties are now complete and can be confirmed with *OK*.

Now the coupler has to be informed with which slaves it should communicate and which data it should exchange with these slaves. For this, highlight the master in the controller configuration tree and press the right mouse button. Select *Append DP Slave* from the appearing context menu. The following window appears:

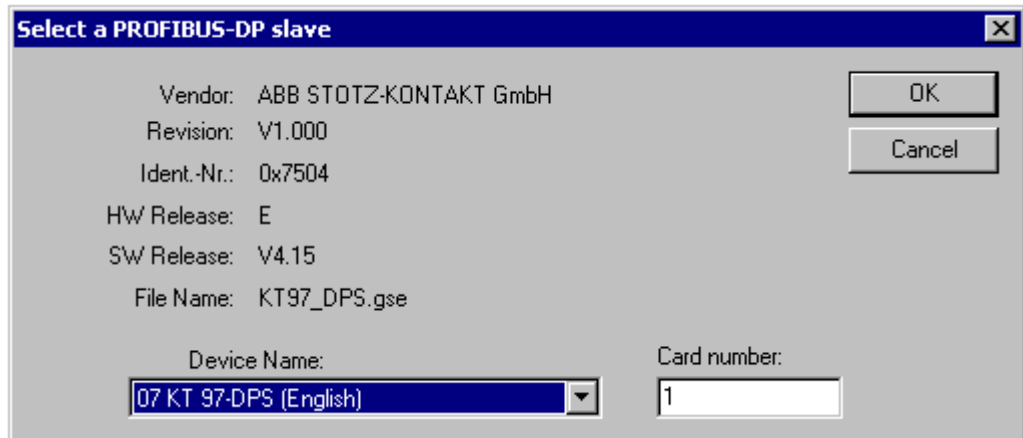


Figure 1-22 Appending a slave

The pull-down menu *Device Name* contains a list of all slaves which are known by the system, i.e. all slaves whose GSD files were available in the subdirectory *PLCCONF* when starting the 907 AC 1131 software (see also chapter Controller configuration in the 907 AC 1131 manual). To add the GSD file of a slave, exit the 907 AC 1131 software, copy the corresponding GSD file to the subdirectory *PLCCONF* and then start the 907 AC 1131 software again. In this example a 07 KT 97 shall be used as DP slave. Therefore, select the entry *07 KT 97-DPS* from the pull-down menu and confirm with *OK*. Repeat the procedure *Append DP Slave* because a valve island (here: FESTO type 02 FB 13) is used as another slave in this example. If you do not have the corresponding GSD file available, insert another 07 KT 97 as DP slave instead.

In the next step the properties of the DP slaves have to be announced to the coupler. For this purpose, select the first slave in the controller configuration tree, press the right mouse button and then select *Properties* from the appearing context menu.

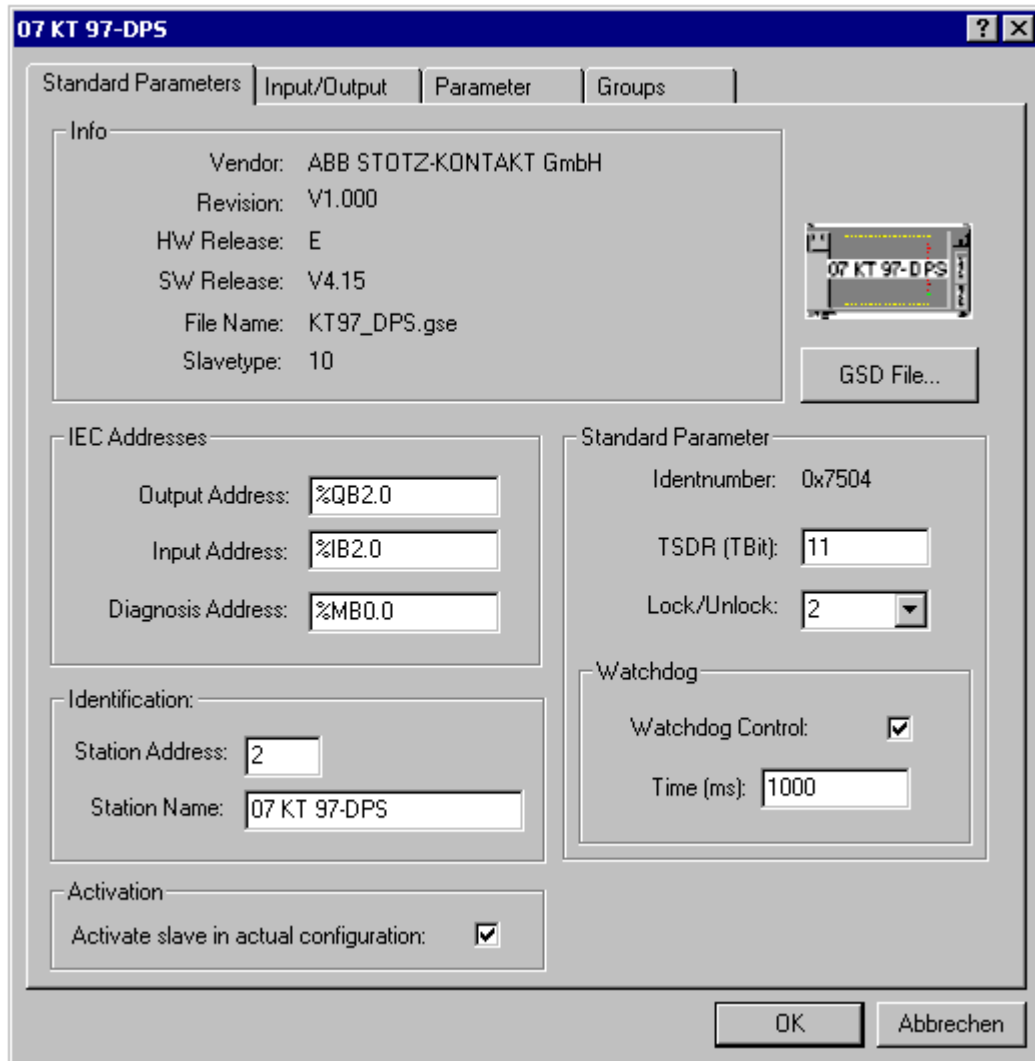


Figure 1-23 Configuration of a slave

For explanations about the individual parameters, refer to the configurator description (see chapter Controller configuration in the 907 AC 1131 documentation). Here, only the bus address assignment and the I/O configuration setting shall be described. According to the defaults, the *Station Address* 2 has to be set for the 07 KT 97 used as slave. In the *Standard Parameters* tab no further settings are required. In order to set the I/O configuration open the *Input/Output* tab.

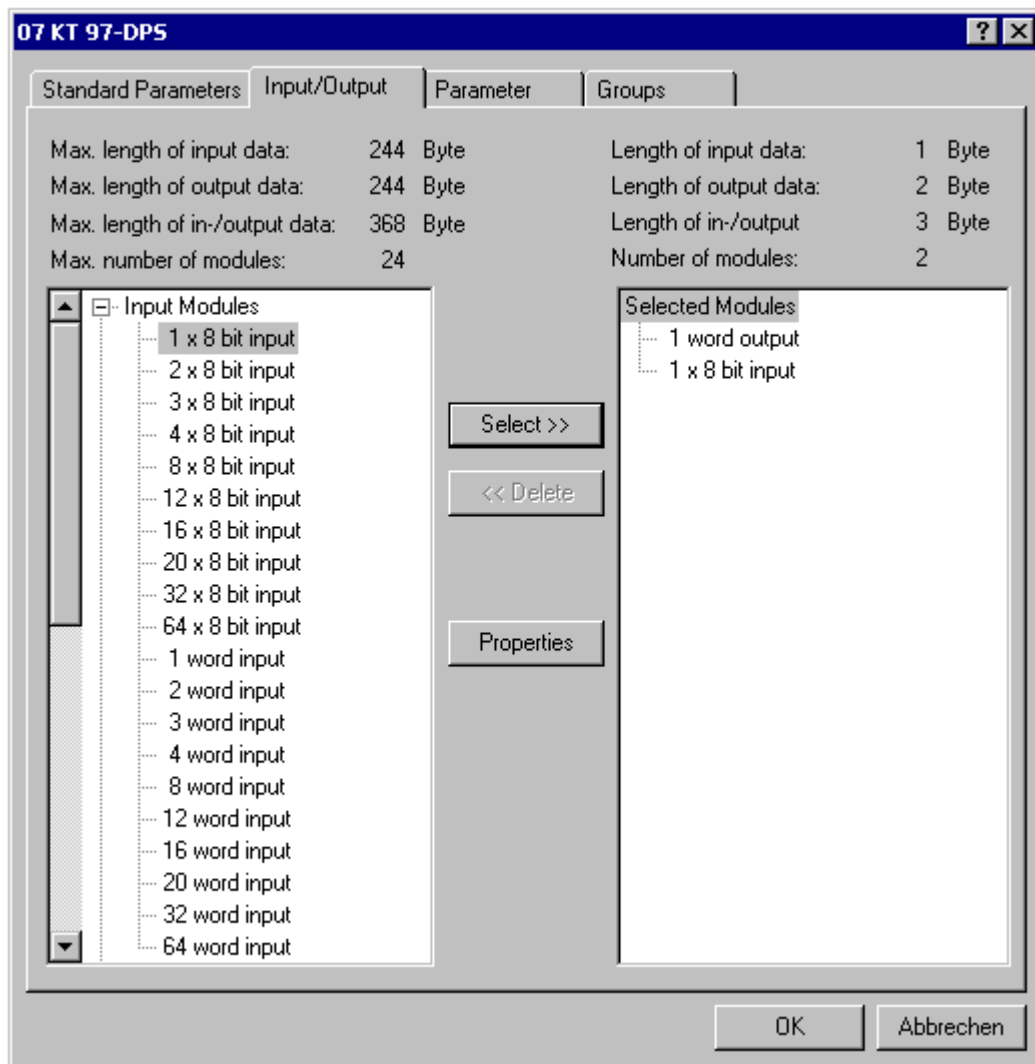


Figure 1-24 I/O configuration of a slave

This tab shows two list areas. On the left-hand side all modules which are available as DP slaves for the 07 KT 97 are listed. On the right-hand side the current configuration is displayed. For the I/O configuration it has to be observed that the PLC is considered as a simple I/O device. The designation of the I/O modules is always performed from the process view. "1 word output" and "1 x 8 bit input" must be selected for the I/O configuration because the master sends one word via PROFIBUS-DP to the slave controller which has to put out this word to the process and receives one byte from the slave which reads this byte from the process. For this, highlight the corresponding module in the list and apply your selection with *Select>>*. In the tabs *Parameters* and *Groups* no settings must be changed (refer to chapter Controller configuration of the 907 AC 1131 manual). Conclusively confirm your configuration with *OK*. The configuration of the first slave is now completed.

Now repeat the configuration procedure for the second slave (valve island or 07 KT 97 as an alternative). According to the definitions made at the beginning, the second slave must be configured with station address 3 and an I/O configuration of "4 valves" (if the valve island is used) or "1 x 8 bit output" (if a second 07 KT 97 is used as an alternative). Furthermore symbolic variable designators for the I/O data can be assigned optionally. Thus, the following hardware configuration results:

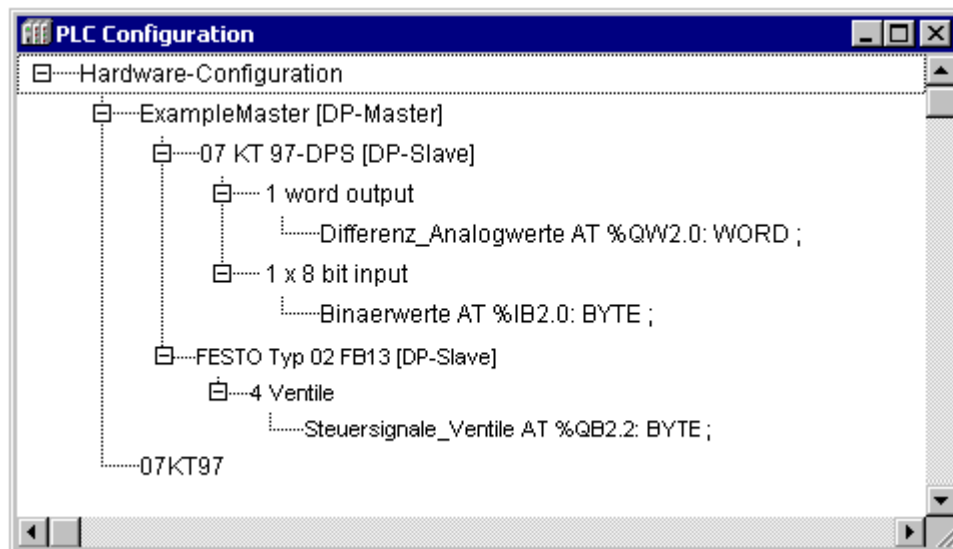



Figure 1-25 Controller configuration of the master

Configuration using 907 FB 1131

Alternatively to the procedure described above the coupler can also be configured using the field bus coupler configuration software 907 FB 1131. For that purpose start the configuration software 907 FB 1131. In the *File* menu select *New*. A window appears listing all available field bus systems. Select *PROFIBUS* and confirm the window with *OK*. The main window now displays an empty bus strand. Save the file under any name to any directory by selecting *File – Save as...* (e.g. *PROFIBUS-Master-Example*). The extension *.pb* (**PROFIBUS**) is automatically attached to the file name.

When configuring a coupler using 907 FB 1131 it is not sufficient to describe only the local coupler. In order to allow an optimal design of the timing behavior the program requires a description of the entire network where the coupler is integrated. Hence, first the 07 KT 97 has to be inserted as the DP master to which the DP slaves and the valve island are assigned in the next step.

Now insert the 07 KT 97 as DP master. For that purpose click on the button *Insert master* . Select *07 KT 97-DPM* in the *Available masters* list and click on the *Add >>* button. After this, the 07 KT 97 is inserted as DP master into the *Selected masters* list. Now specify the desired station address (1 in this example), define a unique description for the device (e.g. *ExampleMaster*) and confirm with *OK*.

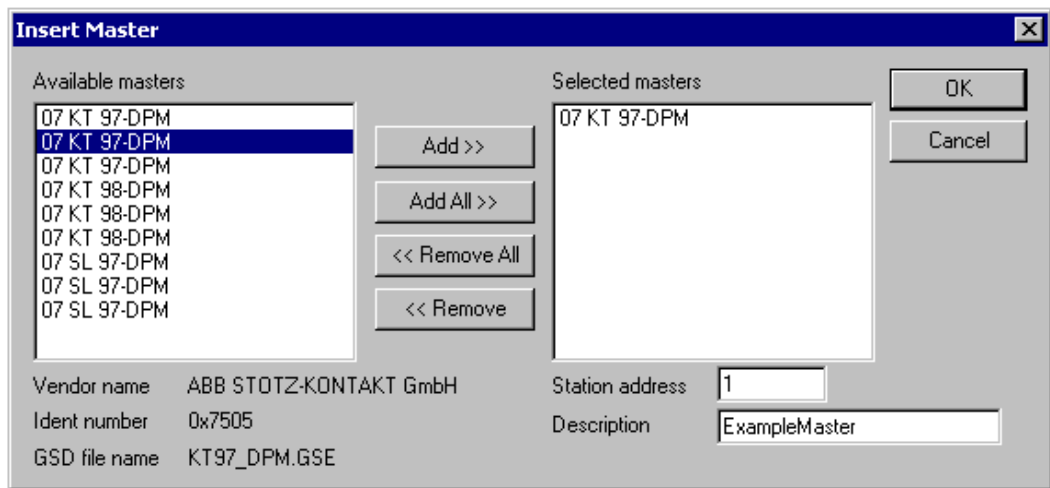



Figure 1-26 Inserting the DP master

Now insert the 07 KT 97 as DP slave in the same way. Select the button *Insert slave*  and then click into a main window area below the previously inserted master. Select *07 KT 97-DPS* in the *Available devices* list and then click on the *Add >>* button. If the desired entry is not visible, set the *Slave filter* for *Vendor* and *Slave Type* to *All*. If, on the other side, the list is confusing because it contains too many entries you can reduce the number of displayed entries by pre-selecting the desired vendor (shown: ABB STOTZ-KONTAKT GmbH) and/or slave type (shown: PLC). After adding, the 07 KT 97-DPS is inserted as DP slave into the *Selected slaves* list. Now specify the desired station address and define a unique device description. For the example given here, enter e.g. station address 2 and the description ExampleSlave according to the definitions made before. Remain in this view in order to insert another slave.

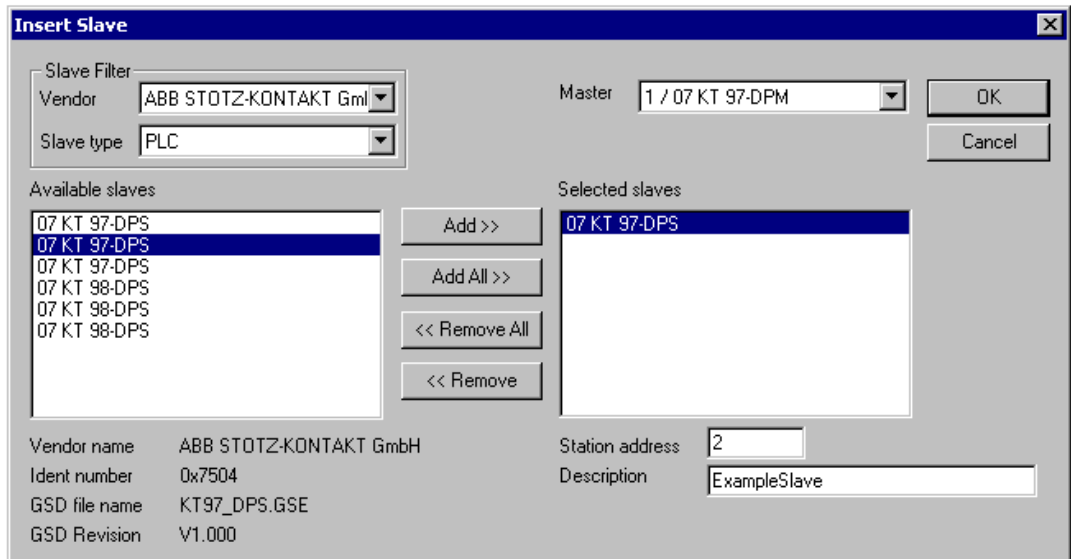


Figure 1-27 Inserting a slave

The further procedure depends on whether the GSD file for the valve island used in this example is available. If this is not the case you can insert another 07 KT 97 as DP slave as an alternative.

If you are alternatively using another 07 KT 97 as slave, highlight the entry *07 KT 97-DPS* in the left-hand list and apply this selection to the current configuration using the *Add>>* button. According to the example, enter station address 3 and the description Valve Island to this slave. Confirm your inputs with *OK*.

To use a device in 907 FB 1131 the software must know the GSD file of the device. If you want to use a device of a foreign manufacturer, you must import the GSD file for this device (refer to 907 FB 1131 documentation). To use the valve island mentioned in this example the corresponding GSD file must be available. First close the currently displayed *Insert Slave* window by clicking on *OK*. Select *File | Copy GSD* from the main menu. Open the directory containing the GSD file. Highlight the file and click on the *Open* button to import the file. In the following dialog you can import the optional bitmap files used for the graphical representation of the device. Select the *Insert slave* button and then click into an empty area below the previously inserted master in the main window in order to insert the valve island as a DP slave as described above. According to the example, enter the station address 3 and the description Valve Island to this slave. Confirm your inputs with *OK*. The resulting network configuration is as follows:

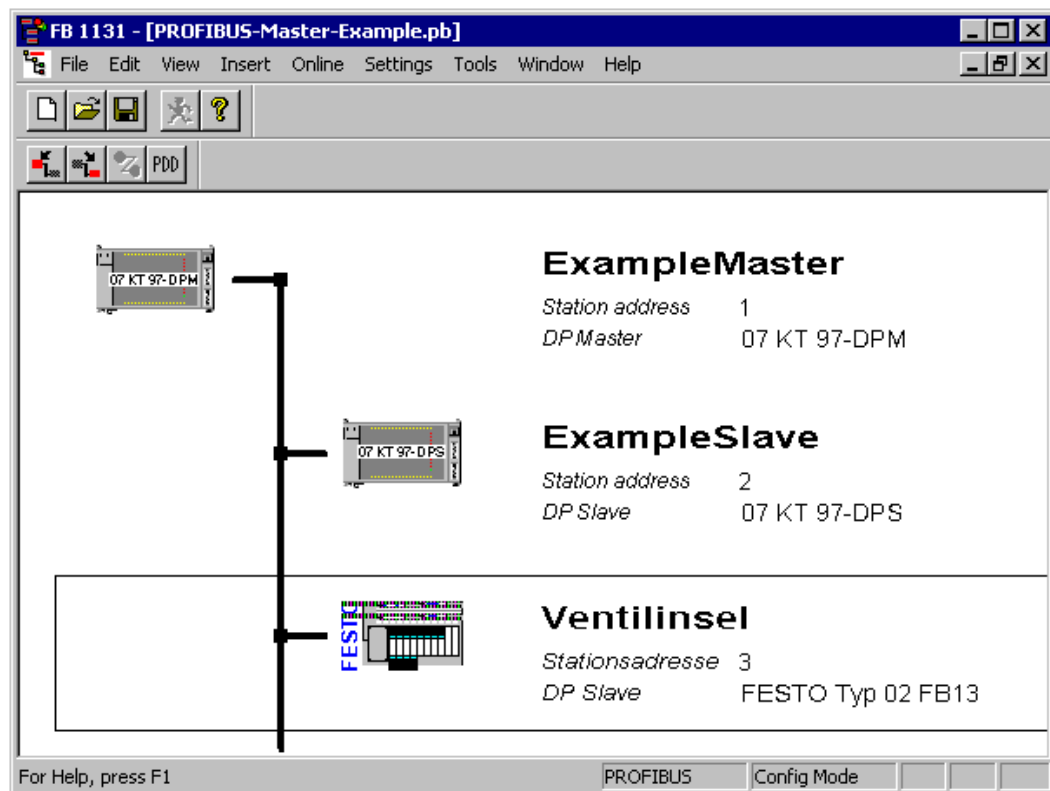


Figure 1-28 Bus overview

In the previous steps the subscribers connected to the bus were described. Now, the communication relationships between these subscribers have to be set (I/O configuration).

For that purpose select the 07 KT 97 which is to be configured as a slave (refer to the figure shown above). In the marked area press the right mouse button and select *Slave configuration* in the appearing context menu. The following window appears.

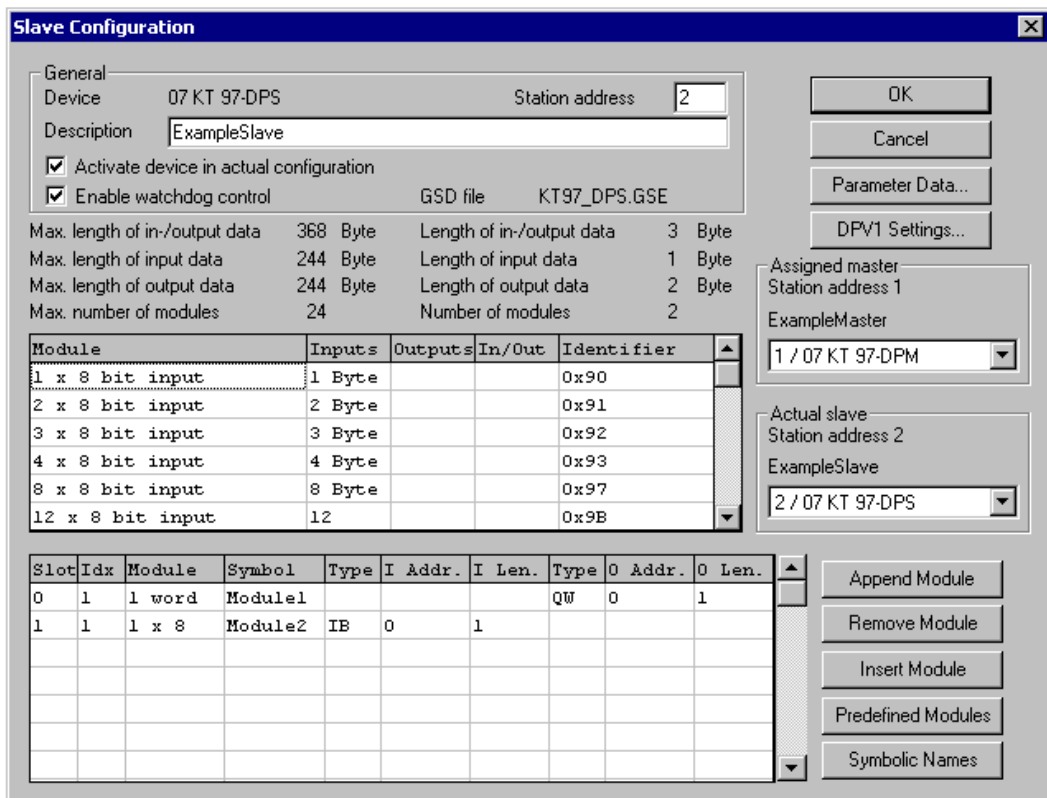


Figure 1-30 Configuration of the 07 KT 97 used as DP slave

Now repeat the configuration procedure for the second slave (valve island or 07 KT 97 as an alternative). According to the definitions made at the beginning, the second slave has to be configured with station address 3 and an I/O configuration of "4 valves" (if the valve island is used) or "1 x 8 bit output" (if a second 07 KT 97 is used as an alternative).

Finally the data transmission rate has to be set. For this, highlight the master in the main window and select *Settings - Bus Parameters*. A window appears displaying a selection of transmission rates which are supported by all subscribers in the network. Select the desired baud rate and confirm with *OK*. The configuration is now completed and should be saved once more.

In order to download the configuration data to the controller, highlight the 07 KT 97 used as DP master and then select *Settings - Device Assignment*. The dialog for configuring the gateway appears (refer to documentation System technology of the basic units S90).

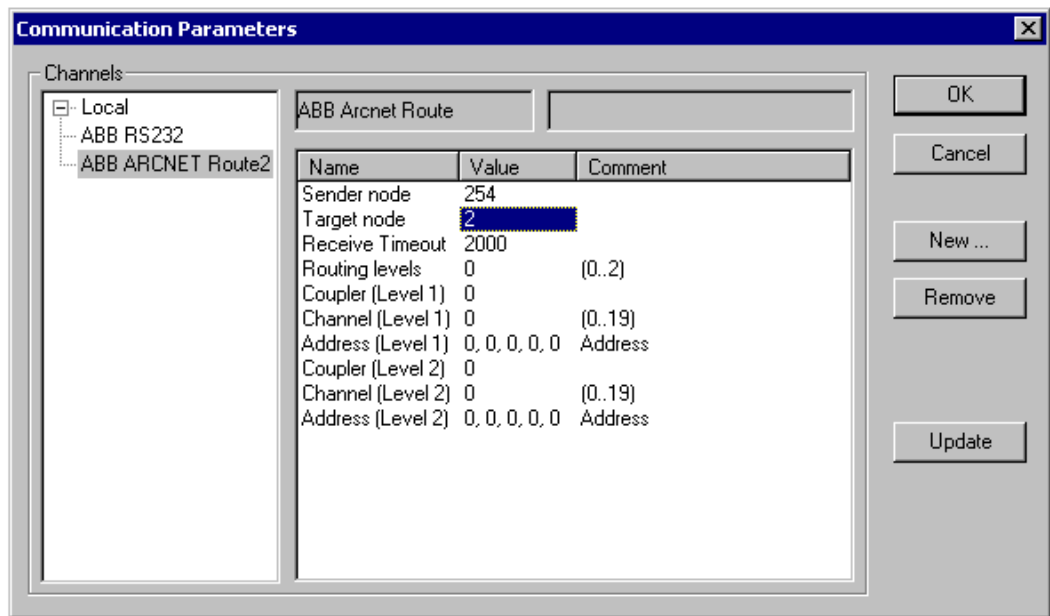


Figure 1-31 Gateway configuration

By selecting *Configure Gateway* you can either open a channel which has already been defined or you can specify a new channel. For transmitting the configuration data the following drivers are available:

- ABB Arcnet Route
- ABB SL97
- ABB RS232 Route

Set the parameters according to the controller to which the data are to be downloaded. Doing this, it is important to specify the correct slot number of the desired coupler. For the controller 07 KT 97 R 162 used in this example, enter slot 2. Select *Connect* to open the set communication channel. If the connection has been established successfully, various information concerning the detected coupler are displayed in the *Device information* area and the field *Errors* contains the value 0. If no communication was possible, the field *Errors* contains a numerical code specifying the occurred error (refer to the documentation for the field bus configurator 907 FB 1131). After the connection has been established successfully, you have to confirm the selection of the coupler by *marking it with a cross* and then clicking on *OK*. Now the previously selected channel is assigned uniquely to the opened configuration file. Select *Online - Download* and confirm the following security dialog with *OK*. A window appears displaying the download progress. The window is closed automatically after the download is completed.

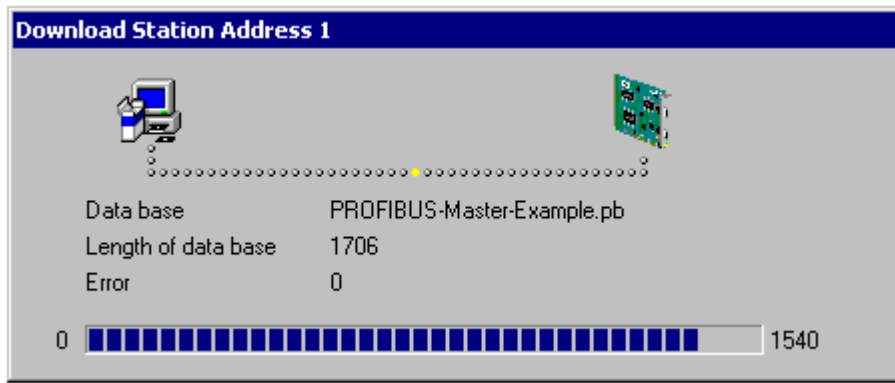


Figure 1-32 Downloading the master

If the configuration data are sent to the coupler without any error, the successful download is additionally indicated by the coupler's green RUN LED which flashes cyclically.

Importing the configuration to 907 AC 1131

Prior to the creation of the actual 907 AC 1131 project, it has to be determined in which areas the DeviceNet I/O data are stored (refer to the documentation of the field bus configurator 907 FB 1131 / section Importing the 907 FB 1131 configuration to 907 AC 1131). Highlight the master and then select *View – Address Table* from the menu. The appearing window displays a list of all slaves communicating with the master and all configured communication connections.

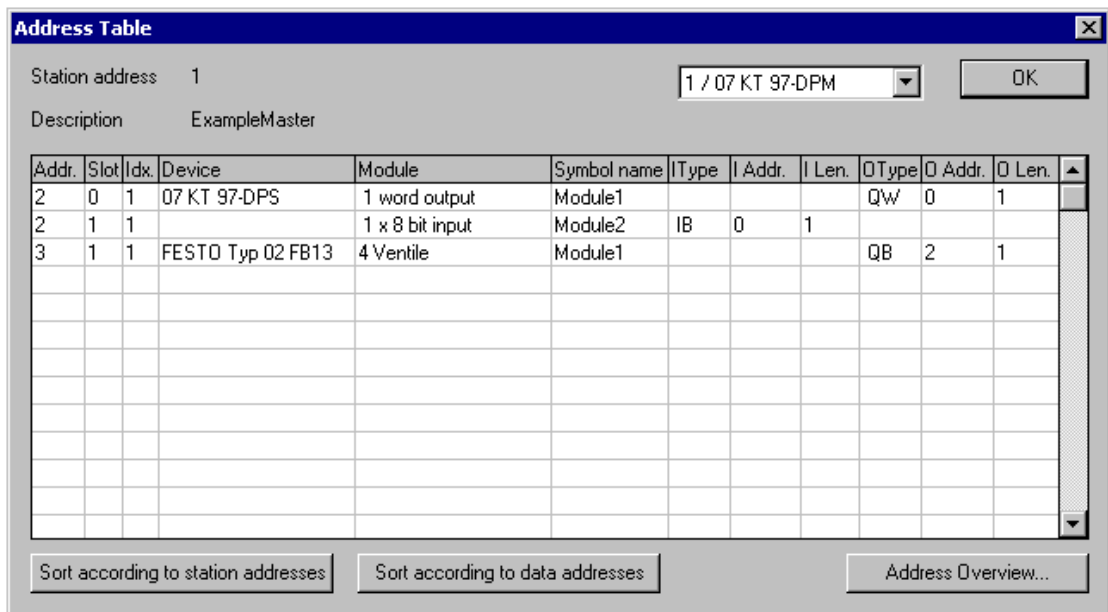


Figure 1-33 Address table

With the help of this table, the appropriate type, size and address of all I/O data has to be determined. These information have to be imported into the related 907 AC 1131 project. For that purpose start the 907 AC 1131 software. Create a new project for a 07 KT 97 controller (refer to 907 AC 1131 manual) and save it as *PROFIBUS-Master-Example*. Select the *Resources* view and create a *New Object* named *FieldbusIOs* in the *Global Variables*.

For each internal field bus coupler (except Arcnet) an own operand area is reserved in the controller (refer to the documentation System technology of the basic units / chapter Operands) where the corresponding coupler stores the data received via the bus (%I area) or where the data which the coupler has to send via the bus (%Q area) must be stored in the user defined program. Apart from the transmission direction (%I, %Q) the areas are identified by the slot

number of the coupler. For the controller 07 KT 97 R 162 used in this example the PROFIBUS coupler is located in slot 2. Thus, the coupler uses the input data range %IX2.? / %IB2.? / %IW2.? and the output data range %QX2.? / %QB2.? / %QW2.?. Within these ranges the actually transmitted data have to be specified. This takes us back to the address table displayed in 907 FB 1131. In this table, an output word (QW) with a length of 1 and an address of 0 is determined for the 07 KT 97 used as slave (addr. 2). Accordingly, in the 907 AC 1131 project a variable of the data type WORD or INT has to be declared within the output operand range of the internal coupler in slot 2 (%QW2.?) with the offset address 0 words (%QW2.0) within this range. Analogous, for the input module of the slave a variable of the data type BYTE has to be declared within the input operand range of the internal coupler in slot 2 (%IB2.?) with an offset address of 0 bytes (%IB2.0) within this range. According to this, for the manipulated variables of the valve island (addr. 3) a variable of the data type BYTE has to be declared within the output operand range of the internal coupler in slot 2 (%QB2.?) with the offset address 0 BYTE (%QB2.0) within this range.

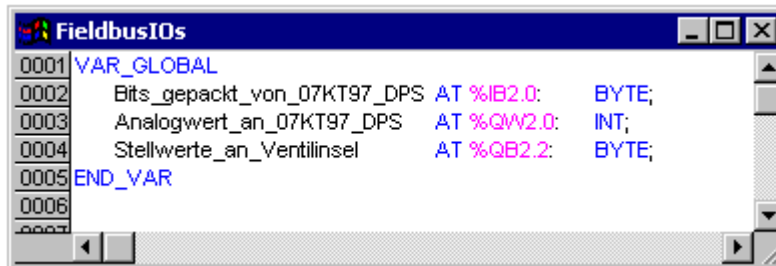
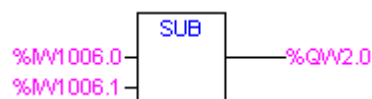


Figure 1-34 Field bus inputs/outputs

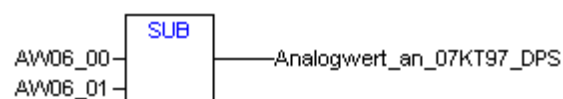
Having done these entries, the PROFIBUS I/Os can be used within the 907 AC 1131 project.

Developing the user program using 907 AC 1131

Now, the actual user program can be created. The calculation of the difference resulting from the analog signals at the inputs EW6,00 and EW6,01 is performed via the corresponding IEC operators



or symbolically (after importing the file KT97_onb.exp)



The control signals for the valve island received by the 07 KT 97 used as DP slave can be immediately forwarded to the valve island using the assignment

Bits_gepackt_von_07KT97_DPS----Stellwerte_an_Ventilinsel

Now the creation of the program for implementing the functionality for this example is completed. Additionally you can insert the function block DPM_STAT to the project in order to read out the coupler status. If necessary, detailed diagnosis information of the DP slaves can be polled using DPM_SLVDIAG function blocks and a survey over the conditions of all slaves in the system can be polled using the function block DPM_SYSDIAG. Now the program can be compiled, downloaded to the controller and tested.

1.7 PROFIBUS-DP diagnosis

1.7.1 Status LEDs

The following figures show the positions of the four status LEDs.

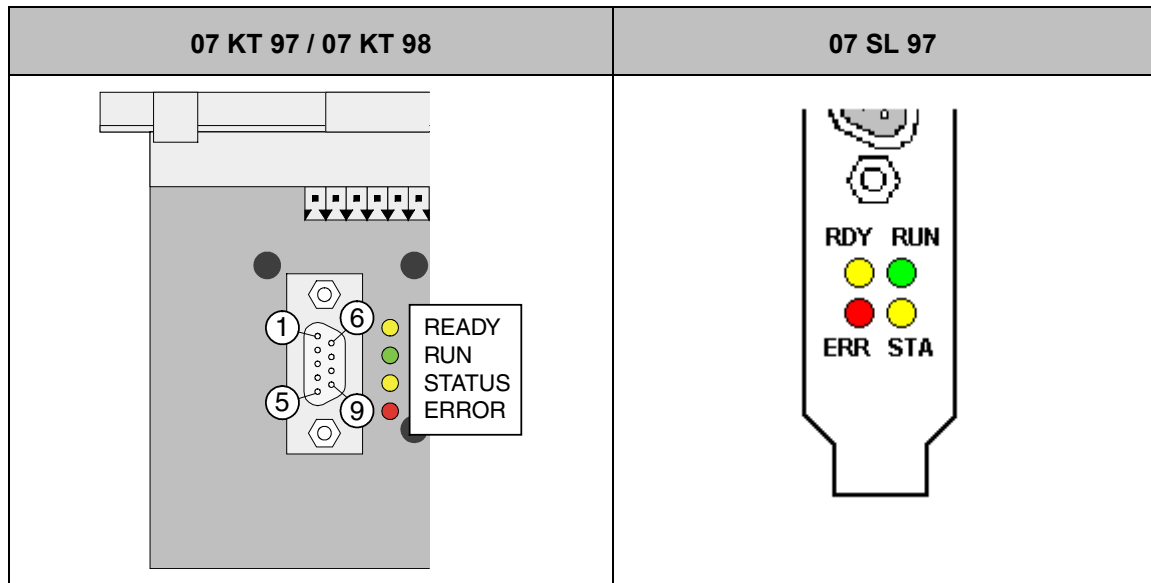


Figure 1-35 Positions of the status LEDs

LED	Color	Status	Meaning
READY	yellow	On flashes cyclic flashes irregularly off	coupler ready bootstrap loader active hardware or system error defective hardware
RUN	green	on flashes cyclic flashes irregularly off	communication is running communication stopped missing or erroneous configuration no communication
STATUS	yellow	on off	DP slave: data exchange with DP master DP master: sending data or token DP slave: no data exchange DP master: no token
ERROR	red	on off	PROFIBUS error no error

Table 1-6 Meanings of the status LEDs

1.7.2 PROFIBUS error messages

The PROFIBUS error messages are listed in section "Error messages of the internal couplers".

1.7.3 Function blocks

PROFIBUS-DP master

- DPM_STAT: Reading the coupler status
- DPM_SLVDIAG: Reading the detailed PROFIBUS diagnosis of a slave
- DPM_SYSDIAG: Reading the system diagnosis

PROFIBUS-DP slave

- DPS_STAT Reading the coupler status

Online diagnosis

The online diagnosis is described in the documentation of the field bus configurator 907 FB 1131.

1.8 Further information

1.8.1 Standardization

EN 50170

DIN 19245 part 1

DIN 19245 part 3

1.8.2 Important addresses

PROFIBUS Nutzerorganisation e.V. (PNO)

Haid-und-Neu-Straße 7

D-76131 Karlsruhe

Phone: (+49) 721 9658 590

Fax: (+49) 721 9658 589

Internet: <http://www.profibus.com>

1.8.3 Terms, definitions and abbreviations used

PROFIBUS-DP	PROCESS FIELDBUS - DECENTRALIZED PERIPHERY
DPM1	DP master (class 1), normal bus master
DPM2	DP master (class 2), commissioning device
DPS	DP slave
GSD	Modules master data
DPV1	Guideline for functional expansions of PROFIBUS-DP
PNO	PROFIBUS user organization (PROFIBUS Nutzer Organisation)

2 Figures

Figure 1-1 Bus terminating resistors connected to the line ends.....	1-5
Figure 1-2 Principle example for a PROFIBUS-DP system with repeaters (1500 kbit baud rate).....	1-6
Figure 1-3 Single master system example.....	1-7
Figure 1-4 Multi master system example.....	1-9
Figure 1-5 Controller configuration in 907 AC 1131.....	1-14
Figure 1-6 Hardware configuration / DP Slave.....	1-15
Figure 1-7 DP slave properties.....	1-15
Figure 1-8 Hardware configuration example 07 KT 97 as DP slave.....	1-16
Figure 1-9 Inserting a master.....	1-17
Figure 1-10 Inserting a slave.....	1-17
Figure 1-11 Bus overview.....	1-18
Figure 1-12 Configuring the coupler as DP slave.....	1-19
Figure 1-13 Completed configuration of the coupler as DP slave.....	1-20
Figure 1-14 Gateway configuration.....	1-20
Figure 1-15 Downloading the configuration to the controller.....	1-21
Figure 1-16 Variables declaration.....	1-22
Figure 1-17 Example for the declaration of global variables.....	1-23
Figure 1-18 PROFIBUS-DP example configuration.....	1-24
Figure 1-19 Controller configuration.....	1-25
Figure 1-20 Inserting the coupler as a master.....	1-25
Figure 1-21 Coupler properties when used as master.....	1-26
Figure 1-22 Appending a slave.....	1-27
Figure 1-23 Configuration of a slave.....	1-28
Figure 1-24 I/O configuration of a slave.....	1-29
Figure 1-25 Controller configuration of the master.....	1-30
Figure 1-26 Inserting the DP master.....	1-31
Figure 1-27 Inserting a slave.....	1-31
Figure 1-28 Bus overview.....	1-32
Figure 1-29 Slave configuration.....	1-33
Figure 1-30 Configuration of the 07 KT 97 used as DP slave.....	1-34
Figure 1-31 Gateway configuration.....	1-35
Figure 1-32 Downloading the master.....	1-36
Figure 1-33 Address table.....	1-36
Figure 1-34 Field bus inputs/outputs.....	1-37
Figure 1-35 Positions of the status LEDs.....	1-38

3 Tables

Table 1-1 Pin assignment of the attachment plug for the bus cable	1-4
Table 1-2 Overview of PROFIBUS libraries	1-12
Table 1-3 Function blocks contained in the library PROFIBUS_S90_V41.LIB	1-12
Table 1-4 Function blocks contained in the library PROFIBUS_Master_S90_V43.LIB	1-13
Table 1-5 Function blocks contained in the library PROFIBUS_Slave_S90_V43.LIB	1-13
Table 1-6 Meanings of the status LEDs	1-38

P

- PROFIBUS-DP brief overview 1-1
- PROFIBUS-DP connection and transfer media 1-4
- PROFIBUS-DP coupler series 90 1-1
- PROFIBUS-DP example DP master 1-24
- PROFIBUS-DP example for DP slave 1-14
- PROFIBUS-DP implementation 1-10
- PROFIBUS-DP planning examples 1-14
- PROFIBUS-DP possibilities for networking 1-7

Content

1	The DeviceNet coupler	C 1-1
1.1	Brief overview	1-1
1.1.1	Fundamental properties and fields of application	1-1
1.1.2	The Communication Model	1-2
1.1.3	Connection Types	1-3
1.1.4	Object Model	1-4
1.1.5	EDS files	1-6
1.1.6	Features	1-6
1.2	Technical data	1-8
1.2.1	Technical data of the coupler	1-8
1.2.2	Interface specification	1-8
1.3	Connection and transfer media	1-8
1.3.1	Attachment plug for the bus cable	1-8
1.3.2	Bus termination resistors	1-9
1.3.3	Bus cable	1-10
1.3.4	Max. lengths of lines	1-11
1.4	Possibilities for networking	1-11
1.4.1	Single master system	1-11
1.4.2	Multi master system	1-13
1.5	DeviceNet implementation	1-14
1.5.1	Configuration	1-14
1.5.2	Running operation	1-14
1.5.3	Error diagnostics	1-14
1.5.4	Function blocks	1-15
1.6	Planning examples	1-15
1.7	Diagnosis	1-24
1.7.1	Status LEDs	1-24
1.7.2	DeviceNet Error messages	1-24
1.7.3	Function blocks	1-24
1.8	Further information	1-25
1.8.1	Standardization	1-25
1.8.2	Important addresses	1-25
1.8.3	Terms, definitions and abbreviations used	1-25
2	Figures	2-1
3	Tables	3-1

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

DeviceNet is mainly used to transfer process data between central controller modules (such as PLC or PC) and decentralized peripheral modules such as I/O modules, drives and valves. To perform this, DeviceNet uses the physics and data transport mechanisms of the Controller Area Network (CAN). The communication is performed message-oriented. The subscribers exchange cyclically, change-controlled or event-controlled data via the logical interconnections specified during the configuration phase. DeviceNet supports transfer rates of 125 kbaud, 250 kbaud and 500 kbaud.

Each of the up to 64 subscribers in a DeviceNet network has an own MAC ID (Media Access Control Identifier, bus address). The Duplicate MAC ID test algorithm, which is executed when booting the system, guarantees that each MAC ID exists only once in the network. DeviceNet does not work by applying the conventional master/slave principle. Each subscriber has transmit rights in principle. As soon as a module reaches a state which activates a planned connection (e.g. change of an input value), it tries to dispatch a corresponding telegram via the bus. In that case it may happen that several subscribers try to send a telegram at the same time. Telegram collisions are detected and removed by the CSMA/CA procedure (Carrier Sense Multiple Access/Collision Avoidance). Each telegram contains a priority identification. The lower this value is, the higher is the priority of the message. The bus address of the sender is part of this priority identification. Due to this, subscribers with a low MAC ID have a higher priority when assigning the bus accesses than subscribers with higher addresses. If a subscriber is sending a telegram and during this time it is interrupted by a telegram with a higher priority, it disconnects from the bus and repeats the transmit procedure after sending the telegram with the higher priority is finished. If a default number of failed transmitting attempts is exceeded, a DeviceNet module changes to the fault state. Due to this, normally low MAC IDs are assigned to DeviceNet masters.

DeviceNet distinguishes between three device types: Client, server and devices which combine both types of functionality. In principle all device types can receive (consume), send (produce) data or perform both operations. A client (master) typically sends data in form of a request and receives data as an answer to this request. Server (slaves) typically consume requests and then produce the answers. Additionally, depending on the planned connection, slaves can send data via the bus without any previous request by a master. Some devices (clients and servers) can only consume messages (pure output devices) or produce them (pure input devices).

Using DeviceNet both mono master and multi master or multi client systems can be realized. Here, each master can access all slaves (servers) theoretically. Prior to commissioning, the actually realized communication connections are determined when configuring the system. The connections are planned in the masters. When booting the system, the master establishes the corresponding connections, i.e. it informs the slaves how the communication between the slaves and the master is performed. Pure servers cannot initiate connections, pure clients cannot receive requests. Only communication connections are possible between a master (client) and several slaves (servers) and such devices, which support both types of functionality. Mixed devices can communicate with other mixed devices as well as with pure servers.

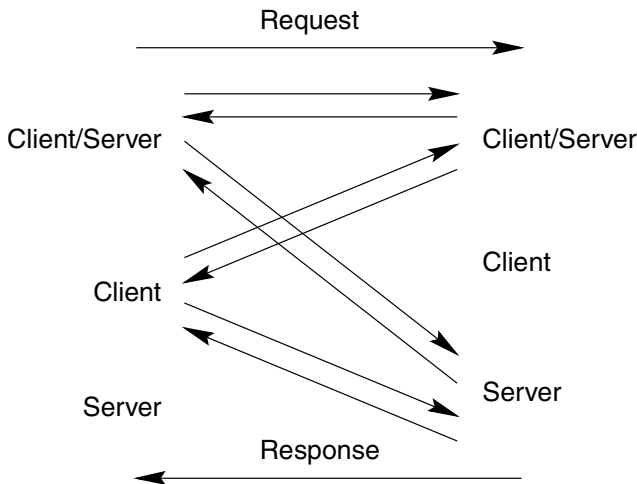


Figure 1-1 Communication connections

1.1.2 The Communication Model

DeviceNet uses the **Producer Consumer Model**. As soon as a DeviceNet subscriber has to transmit data in accordance with the planned connections, it produces data, i.e. it transmits data via the bus. All modules, which are waiting for data, monitor the bus. If a subscriber recognizes that it is the receiver of the data by reading the identifier contained in the telegram header, it consumes the data, i.e. it reads the complete telegram. Here, it is possible that a message can be directed to one single subscriber (single cast) or several subscribers (multi cast).

DeviceNet defines two types of messages: I/O messaging (I/O data transfer) and explicit messaging (direct connection). I/O messaging is used for exchanging time-critical data (e.g. process data). The data are exchanged via single or multi cast connections and typically use identifier with a high priority. The meaning of the messages is determined using the connection IDs (CAN identifier). Before DeviceNet modules can exchange messages, which use these IDs, the modules have to be configured accordingly. The configuration data contain the source and destination address of the object attributes to be transferred for the sender and receiver of the messages. Explicit messaging is a low-priority point to point communication connection between two devices and is typically used for configuration and diagnostic purposes. DeviceNet telegrams (I/O messaging and explicit messaging) contain a maximum of 8 bytes user data. If greater data volumes are to be exchanged, the data have to be partitioned into smaller data packages by fragmenting prior to transmitting. These fragments are then transmitted one after the other and recombined in the receiver.

In addition DeviceNet knows so-called default master slave connections. These connections use special identifiers of the message group 2. In contrast to all other message types these types use the MAC ID as destination address and not as source address. Default connections allow to simplify the communication relations. Here, the connection types bit strobe, polling, change of state and cyclic are available.

1.1.3 Connection Types

For change of state connections (transfer in case of a state change) a DeviceNet subscriber only transmits the data if they have changed since the last transmission. In case of slow processes this can lead to longer transmitting pauses. Due to this all devices are equipped with a heartbeat timer. If this heartbeat timer is elapsed and the data did not change in the mean time, they are transmitted nevertheless. The heartbeat timer is triggered with every new transmitting procedure. That means the subscriber transmits the data in case of a state change, but definitely after the heartbeat interval is elapsed. That way, the receiver is informed that the sender works correctly and did not fail. Hence, transmitting a request to the sender of the data is not required.

Cyclical connections are mainly used if a fixed time slot pattern is predetermined. If, for example, a temperature sensor is located within a slow control system with a sampling rate of 500 ms, it is recommended to update the temperature value cyclically within a pattern of 500 ms. If a higher sampling rate is applied, the controller does not receive additional information and the bus is loaded unnecessarily.

Change of state connections and cyclical connections are acknowledged exchanges per default, i.e. the consumer (receiver) of the data transmits an acknowledgement to the producer (transmitter) as a receipt confirmation. These acknowledgements can be suppressed using the Acknowledge Handler Object (refer to the object model), in order to avoid additional telegram traffic for systems which are anyway heavily loaded due to fast updating times.

In addition multi cast connections are managed using the Acknowledge Handler Object. Not only the master reads slave data, but also other subscribers as for example operating stations monitor the telegram traffic and process as well as acknowledge the data which are required for display, alarm and archive purposes.

In case of polling connections the master transmits a polling command to the slave. Data of the master, which are intended for the slave, are also transferred. This can be performed using one single telegram or with the help of the fragmentation services. Data volumes with a size greater than 8 bytes are divided into subsets and then transferred one after the other. When receiving the data packages transferred by the master, the slave recombines the packages. If the slave contains data, which are intended for the polling master, it transmits the data to the master. The response telegram of the slave contains alternatively or additionally status information. If the slave does not respond to polling requests of the master, a timeout occurs. Especially in case of slow processes polling connections have the disadvantage of an unnecessarily heavy overload of the bus because data do not very often change between two procedures.

Bit strobe connections are used to transfer small data volumes between a master and one or several slaves. The bit strobe telegram transmitted by the master contains 64 bits of user data. Each bit is assigned to one subscriber or bus address. Therefore bit strobe telegrams have a broadcast functionality. As all addressed slaves receive the telegram at the same time, bit strobe is often used for synchronizing input or output data. A slave can put out the assigned bit directly at an output or interpret the bit as a trigger condition, in order to freeze input values and to transfer these values to the master with the next polling telegram. The data send back by the slave are limited to a size of 8 bytes. Therefore bit strobe causes a lower bus load than the polling method.

1.1.4 Object Model

DeviceNet is based on an abstract object model. This model defines the organization and implementation of the attributes (data), the services (methods, procedures) and the behavior of the individual components of a DeviceNet subscriber. The complete definition can be found in the corresponding specification to be obtained from the ODVA. The model provides a four-step addressing scheme for each attribute. The first level is the NodeAddress (MAC ID, bus address), the second one is the Object Class Identifier, the third level is the Instance Number and the last level is the Attribute Number.

Classes:

Each DeviceNet module contains a collection of objects where each object has a certain class. The DeviceNet norm already contains the definition of many standard classes. These standard classes describe for example fundamental properties, the communication behavior or parameters of individual channels of a subscriber. In addition further manufacturer-specific classes can be defined for a DeviceNet module.

The definition of classes can be compared to the definition of STRUCT data types or connection elements in libraries. When defining a structure no data are created at first. Only a possible data format is described. If a connection element library is inserted into a project, the contained connection elements are not executed. The library only contains descriptions of possible connection elements.

Objects and instances:

Each DeviceNet subscriber contains one or several objects of the different classes. An object is an instance of a class. Depending on the type of class only one object or several objects of this class can be available. The instances are numbered continuously.

This can be compared to the creation of variables of STRUCT types which were defined before. In relation to a library this corresponds to the insertion of a connection element into the project available in the library. After a symbolic name has been entered an instance of this type is generated in the project.

Attributes:

Data (attributes) are combined in an object instance. These attributes describe the different properties of a DeviceNet module and can be (partly) read or written by other modules via the bus.

In relation to the STRUCT variables in a project attributes can be compared to the individual elements of a structure. Reading and writing of the individual elements is done by assignments in the program. For connection elements, the inputs and outputs can be seen as attributes.

Standard objects:

The following described standard objects are contained in any DeviceNet module, provided no other information is available in this document. Some information contained in these objects are for example read out automatically by a DeviceNet master from the particular device on commissioning of the bus and are then compared with the required configuration. When booting the system and establishing the communication connections a master writes objects to the assigned slaves. This informs the slave how it has to use the bus.

Each DeviceNet module has an instance of the Identity Object (class identifier 1, instance number 1). Parts of this instance are for example the attributes manufacturer code, device type, product code, version identifier, status, serial number, product name. These information can be read out from a device using the DeviceNet service `Get_Attribute_Single`. The following table shows the structure of the Identity Object.

Identity Object: class identifier 1, instance number 1				
Attribute No.	Attribute name	Data format	Description	Implementation
1	Vendor ID	UINT	Manufacturer identifier	required
2	Device Type	UINT	Device type	required
3	Product Code	UINT	Product number	required
4	Major Revision Minor Revision	USINT USINT	Version identifier	required
5	Status	WORD	General status	required
6	Serial Number	UDINT	Serial number	required
7	Product Name	STRING(32)	Product name	required
8	State	USINT	Current state	optional
9	Configuration Consistency Value	UINT	Device configuration	optional
10	Heartbeat Interval	USINT	Time interval in seconds	optional

Table 1-1 Structure of the Identity Object

In addition DeviceNet modules have an instance of the Message Router Object. The Message Router Object is part of a device which explicitly routes messages to other objects. Usually it is not possible to directly access this object within the DeviceNet network.

The instance of a DeviceNet object of a device contains the attributes (data) bus address (MAC ID), baud rate, bus off reaction, bus off counter, allocation selection and the MAC ID of the master. The access to these data is also performed using the `Get_Attribute_Single` service.

DeviceNet modules have at least one Assembly Object. The main task of these objects is to combine different attributes (data) of different Application Objects into one attribute which can be transmitted in one single message.

In addition DeviceNet modules have at least two Connection Objects. Each Connection Object represents one end point of a virtual connection between two subscribers of a DeviceNet network. These virtual connections are named explicit messaging and I/O messaging (refer to the description above). Explicit messages contain the address and value of an attribute as well as a service identifier which describes how the data are handled. In contrast, an I/O message contains only data. All information about handling the data are available in the Connection Object which is assigned to this message.

The Parameter Object is an optional object and is only used by devices which have adjustable parameters. For each parameter an own instance is available. The Parameter Object provides a standardized access to all parameters for the configuration tools. The configuration options are attributes of the Parameter Object and can be for example a value range or scaling of channels, texts or limits.

Normally at least one Application Object exists in each DeviceNet module in addition to the objects of the assembly or parameter class. At this point these objects are not described in detail. A description can be found in the DeviceNet Object library.

1.1.5 EDS files

The characteristic properties of a DeviceNet module are documented in the form of an electronic data sheet (EDS file, electronic data sheet). The file describes the characteristics (objects) of a module type completely and clearly in a standardized defined and manufacturer independent format. Programs for configuring a DeviceNet network use the module type descriptions available in the EDS files. This strongly simplifies the planning of a DeviceNet system. Usually the EDS files are provided by the module's manufacturer. In addition the ODVA provides many EDS files in the Internet which can be downloaded free of charge.

The Internet address of the ODVA is: <http://www.odva.org>.

1.1.6 Features

Mode of operation:

- DeviceNet master (client)
- Check for double subscriber addresses (MAC IDs) in the bus when initializing the system
- Per slave a maximum of 256 bytes of input data and 256 bytes of output data
- A maximum of 57344 I/O points

Transmission technique:

- ISO 11898, potential separated
- Data lines, (twisted) two wire lines and power supply using only one cable
- Transfer rates of 125 kbit/s, 250 kbit/s and 500 kbit/s
- Bus length up to 500 m at 125 kbit/s and up to 100 m at 500 kbit/s (trunk cable)
- One bus can have up to 64 subscribers
- 5-pin COMBICON socket for bus connection

Communication:

- Message-oriented bus access, CSMA/CA
- Data transfer poll, bit strobe, cyclic or change of state, explicit peer-to-peer (acyclic) via class 2, UCMM group 1, 2 and 3 fragmentation
- Predefined master slave connections
- 8 bytes of unfragmentized user data, for fragmentation any size is possible
- Synchronization of inputs and/or outputs via bit strobe connections

Protection functions:

- Message transfer with hamming distance $HD = 6$.
- Fault recognition mechanisms via 15 bit CRC, frame check, acknowledge, bit monitoring and bit stuffing
- Incorrect parameter settings are avoided because subscribers with faulty parameters are not included in the user data operation
- In case of a subscriber failure the system keeps running. The failure is registered in the master and indicated via a common diagnosis
- Response monitoring of the subscribers (heartbeat) by a cyclical check for double subscriber addresses
- Removing, adding and exchanging of subscribers during running operation is possible

Status indication via 4 LEDs:

- READY (yellow): the coupler is ready for operation
- RUN (green): configuration and communication status
- NET (green/red): network status
- MOD (green/red): module status

1.2 Technical data

1.2.1 Technical data of the coupler

Coupler type	DeviceNet master coupler in PC/104 format
Processor	16 bit processor with interrupt controller and DMA controller
Memory expansion	8 kbytes DP-RAM, 512 kbytes Flash EPROM, 128 kbytes RAM
Internal power supply with	+5 V, 650 mA
Dimensions	96 x 90 x 23 mm
CE sign	55011 Class B for Emissions, EN 50082-2 for Immunity

1.2.2 Interface specification

Interface socket	9-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	DeviceNet, 500 kBaud max.
Transmission rate	Baud-rate 125 kbit/s, 250 kbit/s and 500 kBit/s
Status indication	4 LEDs
Number of subscribers	63 max.

1.3 Connection and transfer media

1.3.1 Attachment plug for the bus cable

5-pin COMBICON connector

Assignment:

Pin No.	typical core color	Signal	Meaning
1	Black	-V	Reference potential for external power supply +24 V
2	Blue	CANL	Receive/transmit line, low
3	Blank	Shield	Shield of the bus line
4	White	CANH	Receive/transmit line, high
5	Red	+V	+24 V external power supply

Table 1-2 Assignment

It is absolutely necessary that all cables (i.e. the data lines CANH / CANL, the external 24 V power supply +V / -V and the shielding) are connected.

Supplier:

e. g. COMBICON
Phoenix Contact GmbH & Co.
Flachsmarktstraße 8 - 28
D-32825 Blomberg
Telephone: (+49) 52 35 / 3-00
Telefax: (+49) 52 35 / 3-4 12 00
Internet: <http://www.phoenixcontact.com>

1.3.2 Bus termination resistors

The endings of the data lines (of the bus segments) have to be terminated with a 120 Ω bus termination resistor. The bus termination resistor is usually installed directly at the bus connector.

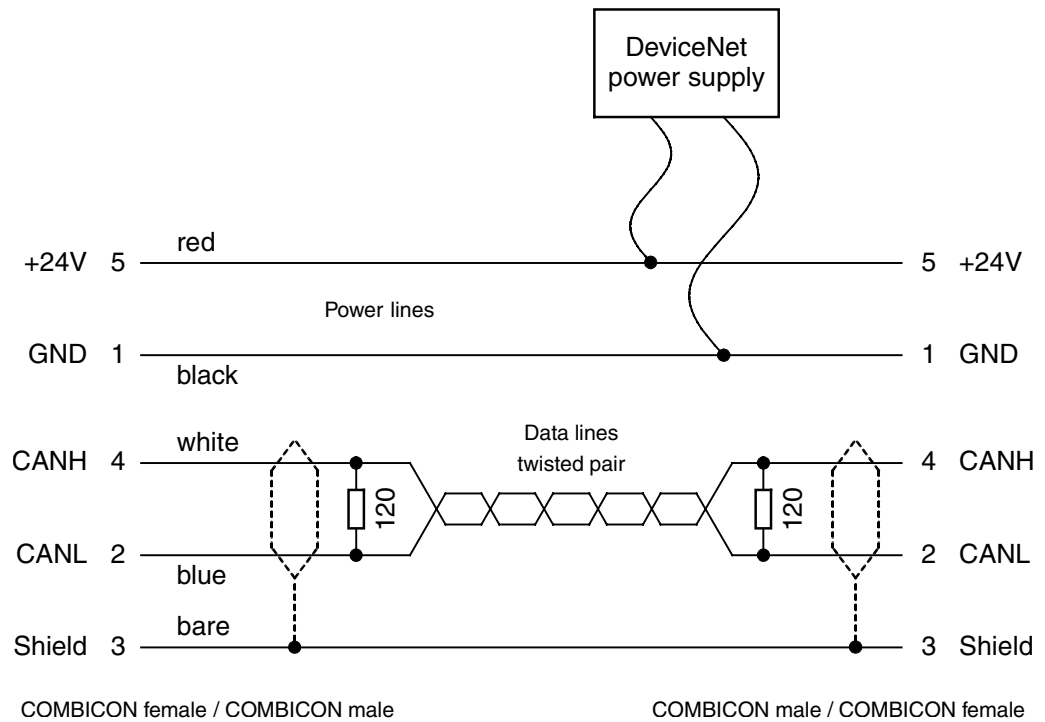


Figure 1-2 Bus termination resistors

1.3.3 Bus cable

Two cable types are used with DeviceNet: Trunk Cable and Drop Cable. The Trunk Cable has a greater line diameter than the Drop Cable. In principle both line types can be used as main lines (Trunk Line) and as branch lines (Drop Line). However, using the thinner line reduces the maximum line lengths.

Trunk Cable (main lines):

Data lines:

Type:	Twisted pair cable (shielded)
Wave impedance (cable impedance):	120 Ω
Cable capacity (distributed capacitance):	≤ 40 nF/km
Line cross section:	≥ 1.0 mm ² (18 AWG/19)
Line resistance per core:	≤ 22.5 Ω /km
Loop resistance (resistance of 2 cores):	≤ 45 Ω /km

Power supply:

Loop resistance (resistance of 2 cores):	≤ 45 Ω /km
Line cross section:	≥ 1.5 mm ² (15 AWG/19)

Drop Cable (branch lines):

Data lines:

Type:	Twisted pair cable (shielded)
Wave impedance (cable impedance):	120 Ω
Cable capacity (distributed capacitance):	≤ 40 nF/km
Line cross section:	≥ 0.25 mm ² (24 AWG/19)
Line resistance per core:	≤ 92 Ω /km
Loop resistance (resistance of 2 cores):	≤ 184 Ω /km

Power supply:

Loop resistance (resistance of 2 cores):	≤ 45 Ω /km
Line cross section:	≥ 0.34 mm ² (22 AWG/19)

Supplier:

e.g. UNITRONIC® BUS DeviceNet

U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
D-70565 Stuttgart
Telephone: (+49) 711 7838 01
Telefax: (+49) 711 7838 264
Internet: <http://www.lappkabel.de>

1.3.4 Max. lengths of lines

Transmission rate	125 Kbit/s	250 Kbit/s	500 Kbit/s
max. length of main strand Trunk Cable	500 m (1.610 ft)	250 m (820 ft)	100 m (328 ft)
max. length of main strand Drop Cable	100 m (328 ft)	100 m (328 ft)	100 m (328 ft)
max. length for each branch line Trunk Cable / Drop Cable	6 m (20 ft)	6 m (20 ft)	6 m (20 ft)
max. line length: sum of branch lines Trunk Cable / Drop Cable	156 m (512 ft)	78 m (256 ft)	39 m (128 ft)

Table 1-3 Maximum line lengths

1.4 Possibilities for networking

The DeviceNet coupler is connected to the bus via the 5-pin COMBICON socket. As an EMC measure and for protection against dangerous contact voltages, the shield of the bus line must be connected to protective earth outside the housing.

1.4.1 Single master system

The single master system is the simplest version of a DeviceNet network. It consists of a DeviceNet master (Client) and up to 63 slaves (Server). The line ends of the busses have to be terminated using bus termination resistors.

The DeviceNet master is able to:

- automatically determine the structure of the system, i.e. the description and configuration possibilities of all subscribers
- detect doubled bus addresses in the network
- parameterize slaves (e.g. communication connections, time supervision, bus traffic)
- configure slaves (e.g. type, number and operating mode of channels)
- read input data of the slaves
- write output data of the slaves
- read diagnostic data of the slaves
- send control commands to the slaves (via bit strobe, e.g. freezing input signals)
- read, write and reset slave objects even if the slaves are in running mode

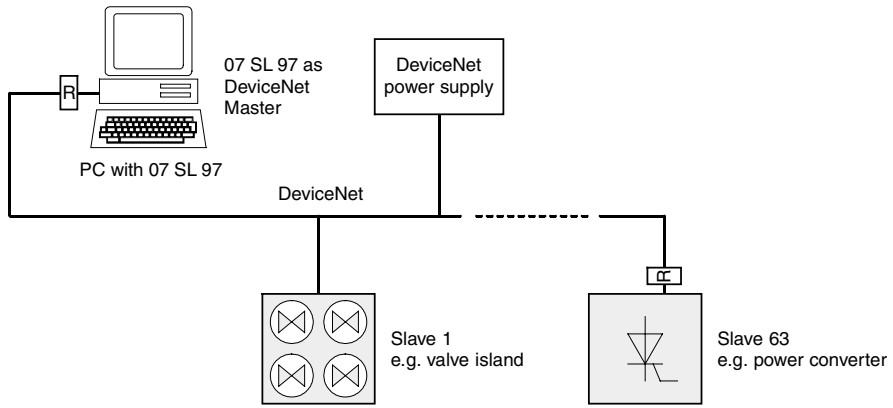


Figure 1-3 Example of a single master system

1.4.2 Multi master system

A DeviceNet network containing several masters is called a multi master system. Up to 64 subscribers (master and slaves) can be operated in this network type. In a multi master system the master-slave communication is possible as well as the data exchange between masters.

Masters are able to

- automatically determine the structure of the system, i.e. the description and configuration possibilities of all subscribers
- parameterize slaves (e.g. communication connections, time supervision, bus traffic)
- configure slaves (e.g. type, number and operating mode of channels)
- read input data of the slaves
- write output data of the slaves
- read diagnostic data of the slaves
- send control commands to the slaves (via bit strobe, e.g. freezing input signals)
- read, write and reset slave objects even if the slaves are in running mode

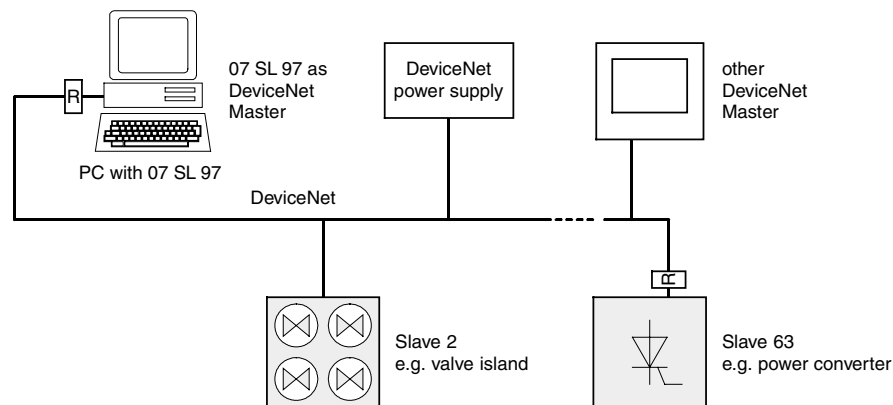


Figure 1-4 Example of a multi master system

1.5 DeviceNet implementation

1.5.1 Configuration

The DeviceNet master coupler is configured via the PC using the configuration software for field bus couplers 907 FB 1131 (see documentation 907 FC 1131 and project engineering example). Configuration data created using 907 FB 1131 are not assigned directly to a project. Thus, the resulting file has to be send separately to the control (additionally to a 907 AC 1131 project). It is there stored in a Flash memory automatically.

If the user defined project is written to SMC, also the configuration data are automatically saved.

1.5.2 Running operation

The DeviceNet protocol is automatically processed by the coupler and the operating system of the control. The coupler is only active on the bus if it was correctly initialized before and if the user program runs. No function blocks are necessary for exchanging process data via DeviceNet. Special DeviceNet functions can be realized using the function blocks of the DeviceNet library (see library Devnet + Coupler).

The coupler starts the communication via DeviceNet after the user defined program is started and attempts to initialize the planned slaves. After a slave is initialized successfully it is added to the process data exchange. The I/O data exchange with the slaves is performed automatically.

If the user program is stopped, the coupler shuts down the DeviceNet communication in a controlled manner.

1.5.3 Error diagnostics

DeviceNet communication errors are always indicated by the LEDs beside the bus connector. A malfunction of the DeviceNet driver or the coupler itself is indicated via the FK error flags and the corresponding LEDs (see error tables). Furthermore the DeviceNet library provides various function blocks which allow a detailed error diagnosis (see library DevNet + Coupler).

1.5.4 Function blocks

Libraries:

- **DeviceNet_Master_S90_Vxx.LIB and Coupler_S90_Vxx.LIB**

General:

- DEVNET_INFO – Reading of coupler information

DeviceNet master:

Status / Diagnosis

- DNM_DEVDIAG – Reading the detailed diagnosis of a slave
- DNM_STAT – Reading the DeviceNet coupler status
- DNM_SYSDIAG – Displaying status information of all slaves

Object and attribute handling:

- DNM_GET_ATTR – Reading an attribute from an object in a slave
- DNM_RESET_OBJ – Resetting an object in a slave
- DNM_SET_ATTR – Writing an attribute to an object in a slave

1.6 Planning examples

Below, a planning example for the 07 SL 97 R 165 used as DeviceNet master is shown. Beneath the DeviceNet master 07 SL 97, a valve and a temperature sensor are available operating as DeviceNet slaves in the network. So the resulting configuration is as follows:

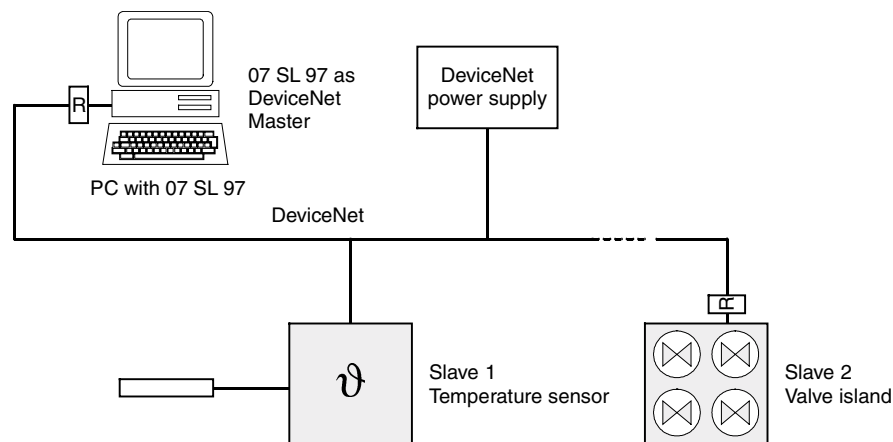


Figure 1-5 Planning example

The control used as DeviceNet master reads the temperature values measured by the sensor. Depending on this value, a control signal is generated and sent to the valve. The following MAC-IDs are defined: 07 SL 97 DeviceNet master 0, temperature sensor 1, valve 2.


Configuration with 907 FB 1131

Prior to taking the master PLC into operation it makes sense to configure the slaves first. Refer to the manufacturer's documentation for the configuration procedure of the sensor and the valve. Such devices normally have DIP switches or rotary switches used to set the bus address (MAC-ID) as well as the transmission rate (if not determined automatically). The I/O configuration is usually fixed.

After the DeviceNet slaves have been configured, the configuration of the master can be done. For that purpose start the configuration software 907 FB 1131. If the slaves detailed in this example are used the first time, you first have to import the EDS files (refer to 907 FB 1131 documentation). In our example, this is not required, because we are alternatively using the provided default ESD file.

In the menu *File* select *New*. A window appears listing all available field bus systems. In this window select *DeviceNet* and confirm with *OK*. The main window now displays an empty bus strand. Save the file under any name in any directory by selecting *File – Save as...* (e.g. *DeviceNet_Master_Sample*). The file name automatically gets the extension *.dn (Device Net)*.

Now insert the 07 SL 97 as DeviceNet master. For that purpose click on the button *Insert*

. Select *07 SL 97-DNM* in the *Available masters* list and click on the button *Add >>*. After this, the *07 SL 97* is inserted as DeviceNet master into the *Selected masters* list. Now you have to specify the desired MAC-ID (0 in this example), define a unique description for the device (e.g. *sample_master*) and confirm with *OK*.

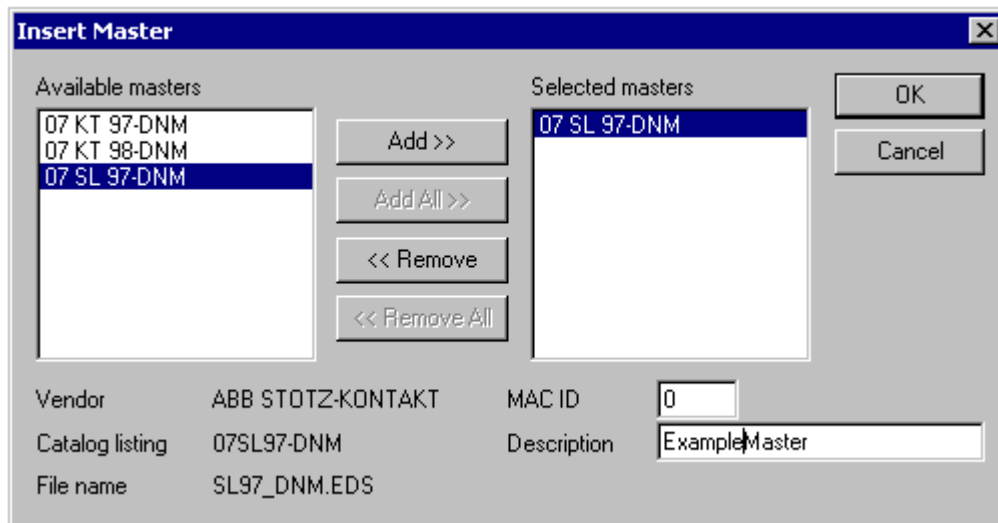



Figure 1-6 Inserting a master

Now insert the slaves in the same way. Select the button *Insert device*  and click into an area below the previously inserted master in the main window. Select *I/O system* in the *Available devices* list and click on the button *Add >>*. If the desired entry is not visible, set the *Device filter* for *Vendor* and *Type* to *All*. After adding, the *I/O system* is inserted as DeviceNet slave into the *Selected devices* list. Now specify the desired MAC-ID and define a unique device description. As this device represents the temperature sensor in our example, enter the MAC-ID 1 and for example the description *Temperature sensor*. Change again to the list containing the available devices again in order to select the second slave. Select once again the entry *I/O system* and apply your selection with *Add>>*. For this slave set the MAC-ID 2 and enter the description *Valve*. Confirm with *OK*.

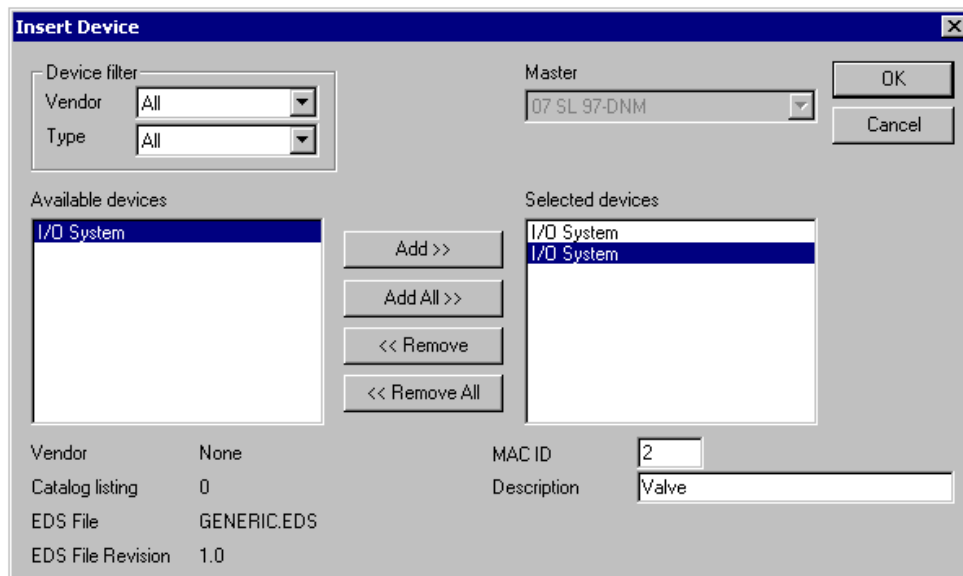


Figure 1-7 Inserting a device

Thus, the following network configuration is obtained:

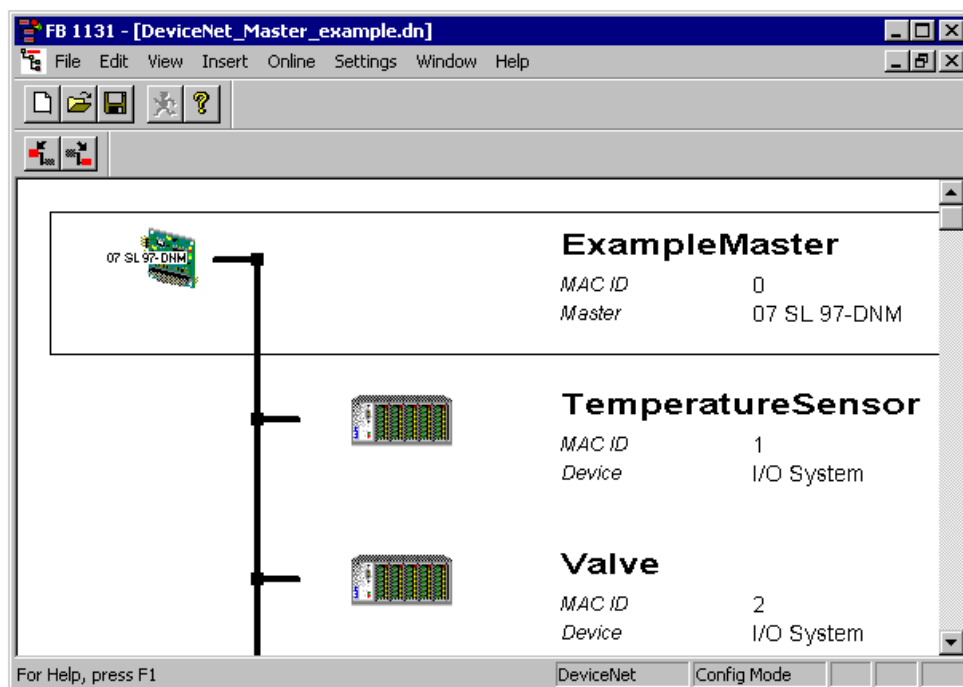


Figure 1-8 Bus overview

In the previous steps the subscribers connected to the bus were described. Now, the communication relationships between these subscribers have to be set (I/O configuration).

For that purpose first select the temperature sensor (refer to the figure shown above). In the marked area press the right mouse button and select *Device configuration* in the appearing context menu. The following window appears.

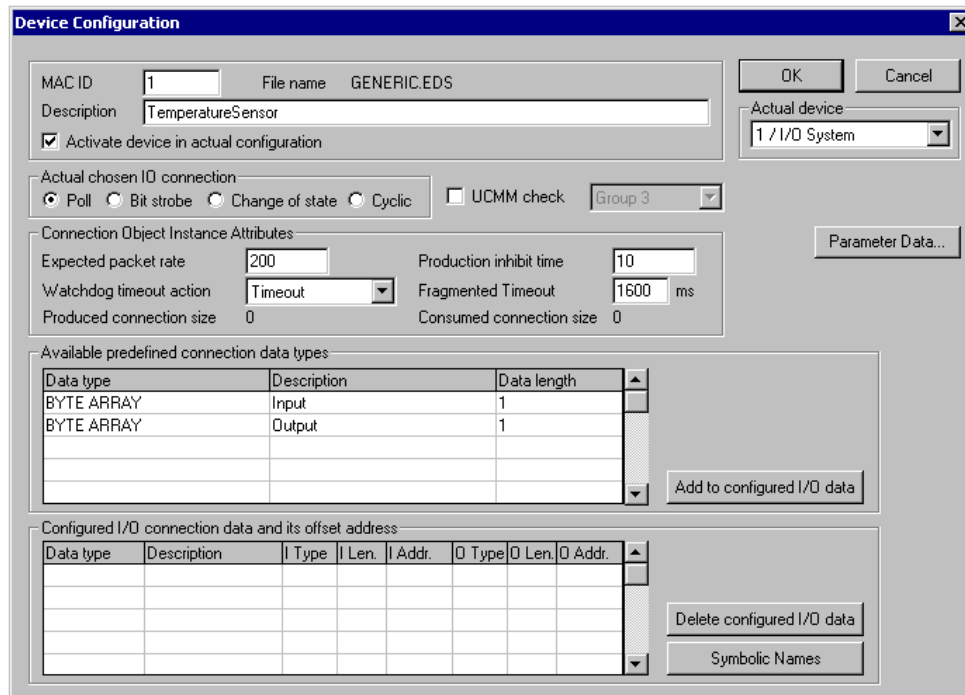


Figure 1-9 Device Configuration window

In this view the communication relationships between the selected slave and the master are defined. It is possible to establish several parallel relationships. In this example the master should only read the current temperature from the slave. Thus, only one single I/O connection is required. Assuming that the measured temperature is changing very slowly, it is sufficient to read the value periodically in an interval of one second. For this, mark the item *cyclic* in the selection *Currently selected I/O connection* and enter 1000 (1000 ms = 1 s) as value for the cycle time *Expected Packet Rate*. In our example the further possible settings are not described in detail. For their meanings please refer to the 907 FB 1131 documentation. By performing the previous steps you have specified in which way the data are transmitted. Now you have to define, which data should be exchanged in this way. The list *Available predefined connection data types* displays all I/O data types which are supported by the device to be configured. Note, that the designation of the transmission direction (Input/Output) always reflects the view of the process. This means that input data are data which a remote slave either receives from the process and sends to the master or which the master receives from a slave. "Output" designates data which a master sends to a slave or which a slave receives from a master and which the remote slave outputs to the process. Assuming that the slave detects the temperature from the process as a byte value and transmits this byte to the master, you have to select the entry *BYTE ARRAY | Input* in this list and apply the selection with *ADD*. The selected data type is then displayed in the lower list *Configured Connection Data with Offset Addresses*. Since only one byte is transmitted, you can apply this entry. Otherwise the entry in the *E Län.* column has to be edited accordingly. The I/O configuration of the temperature sensor is now completed and can be confirmed with *OK*.

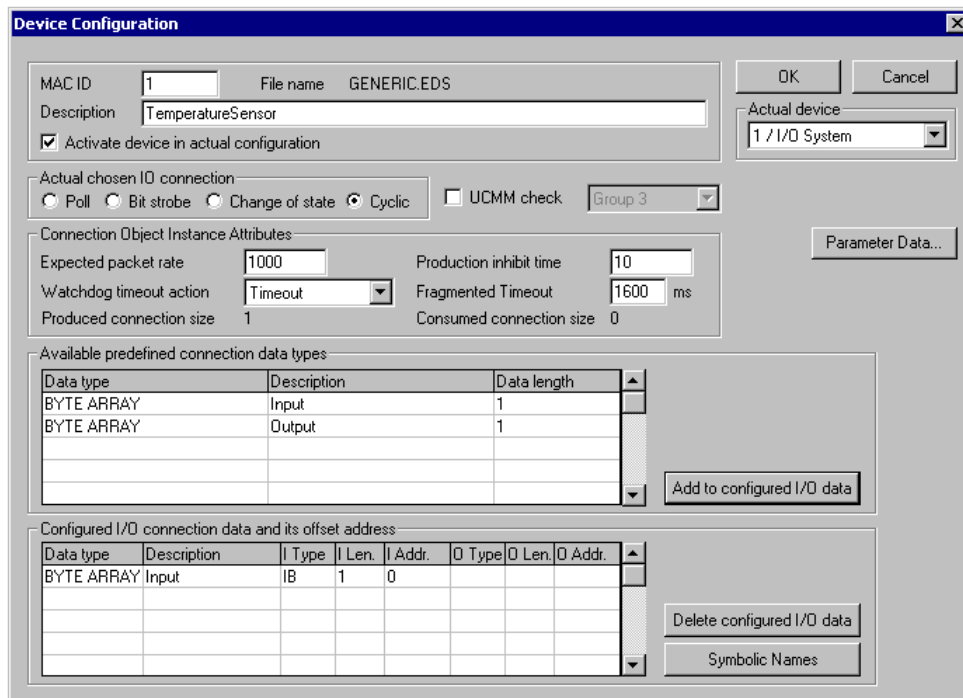


Figure 1-10 Configuration of the temperature sensor

It was specified, that the sensor transmits the measured temperature – available in the data type byte – periodically to the master, scheduled by a time interval of one second.

In the next step the valve has to be configured accordingly as second slave. The valve obtains its manipulated variable as a byte value via the bus and passes it to the process. To avoid an unnecessary bus load, it is specified, that the value should only be transmitted, if it has changed since the last transmission (*Change of state*), but at least after 30 s (*Expected Packet Rate* = 30000 ms). This results in the following DeviceNet slave configuration for the valve.

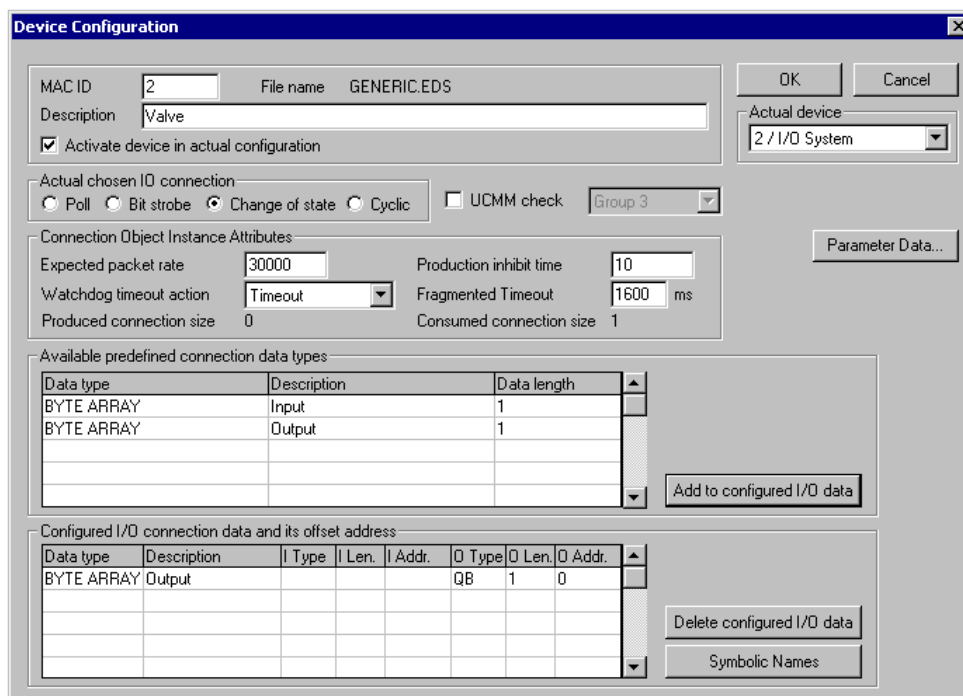


Figure 1-11 Valve configuration

Finally the data transmission rate has to be set. For this, mark the master in the main window and select *Settings – Bus Parameter*. A window appears displaying a selection of transmission rates which are supported by subscribers in the network. Select the desired baud rate and confirm with *OK*. The configuration is now completed and should be saved once more.

In order to download the configuration data to the control, mark the master and select *Settings – Device Assignment*. A dialog appears used for configuring the gateway (Reference to document gateway-configuration).

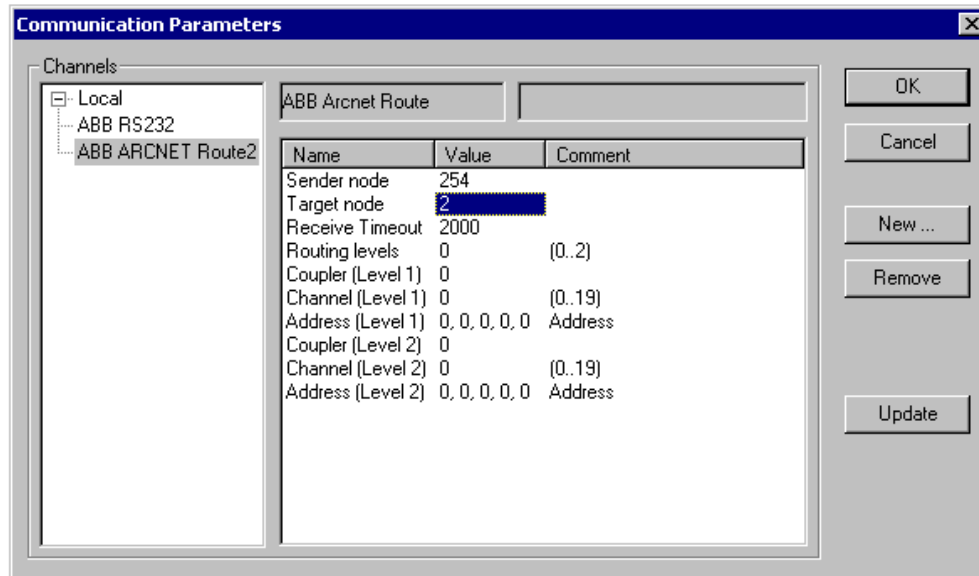


Figure 1-12 Gateway configuration

By selecting *Configure Gateway* you can either open a channel which has already been defined or you can specify a new channel. For transmitting the configuration data the following drivers are available:

- ABB Arcnet Route
- ABB SL97
- ABB RS232 Route

Set the parameters according to the control to which the data have to be downloaded. Doing this, it is important to specify the correct slot number of the desired coupler. For the control 07 SL 97 R 165 used in this example, enter slot 2. Select *Connect* to open the set communication channel. If the connection has been established successfully, various information concerning the detected coupler are displayed in the *Device information* area and the field *Errors* contains the value 0. If no communication was possible, the field *Errors* contains a numerical code specifying the occurred error (refer to the 907 FB 1131 documentation). After the connection has been established successfully, you have to confirm the selection of the coupler by *marking it with a cross* and then clicking on *OK*. Now the previously selected channel is assigned uniquely to the opened configuration file. Now select *Online – Download* and confirm the following security dialog with *OK*. A window appears displaying the download progress. The window is closed automatically after the download is completed.

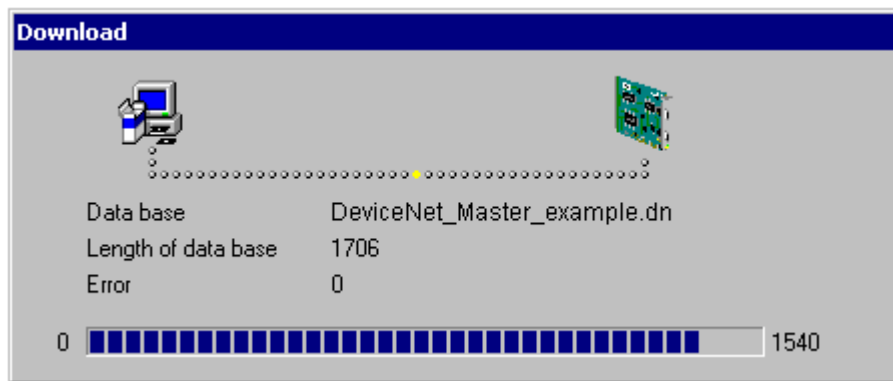


Figure 1-13 Downloading the configuration

If the configuration data are sent to the coupler without any error, the success is additionally indicated by the coupler's green RUN LED which flashes cyclically.

Importing the configuration to 907 AC 1131

Prior to the creation of the actual 907 AC 1131 project, it has to be determined in which areas the DeviceNet I/O data should be stored (Reference to section 907 FB 1131 documentation importing configuration data in 907 AC 1131). Mark the master and select *View – Address Table*. The appearing window displays a list with all slaves communicating with the master and all configured communication connections.

MAC ID	Device	Module	Com.	I Type	I Len.	I Addr.	O Type	O Len.	O Addr.
1	I/O System	Input	Cyclic	IB	1	0			
2	I/O System	Output	CDS				QB	1	0

Figure 1-14 Address table

With the help of this table, the appropriate type, size and address of all I/O data have to be determined. These information have to be imported into the related 907 AC 1131 project. For that purpose start 907 AC 1131. Create a new project for the control 07 SL 97 (see 907 AC 1131 documentation) and save it as *DeviceNet_Master_Example*. Select the *Resources* view and create a *New Object* named *FieldbusIOs* in the *Global Variables*.

For each fieldbus coupler (except Arcnet) an own operand range is reserved in the control (Reference to documentation *System technology / Operands*). In this area the corresponding coupler stores data received via the bus (%I area) or the data are written in the user defined program which the coupler has to send via the bus (%Q area). Beneath the transmission direction (%I, %Q) the areas are identified by the slot number of the coupler. For the control 07 SL 97 R 165 used in this example the DeviceNet master coupler is located in slot 2. Thus,

the coupler uses the input data range %IX2.? / %IB2.? / %IW2.? and the output data range %QX2.? / %QB2.? / %QW2.?. Within these ranges the actually transmitted data have to be specified. This takes us back to the address table displayed in 907 FB 1131. For the temperature sensor (MAC ID 1) an InputByte (IB) with the size 1 and address 0 is entered in this table. Accordingly, a variable of the data type BYTE has to be declared in the 907 AC 1131 project which is located within the input operand range of the internal coupler in slot 2 (%IB2.?). The variable has the offset address 0 BYTE (%IB2.0) within this range. Analogous, a variable of the data type BYTE has to be declared for the manipulated variable of the valve (MAC ID 2) which is located within the output operand range of the internal coupler in slot 2 (%QB2.?). The variable has the offset address 0 BYTE (%QB2.0) within this range.

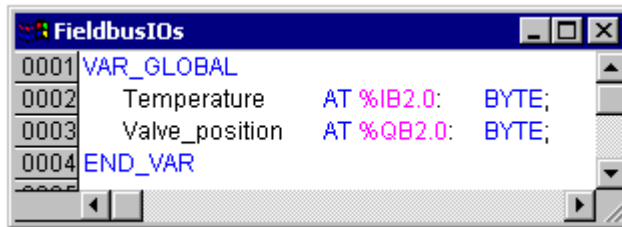


Figure 1-15 Operands

Having done these entries, the DeviceNet I/Os can be used within the 907 AC 1131 project.

Developing the PLC program using 907 AC 1131

Now, the actual user program can be edited. The program is intended to close the valve if the temperature is below 55 °C (manipulated variable = 0(%)) and to open it completely if a temperature higher than 55 °C is measured (manipulated variable = 100(%)). This functionality can be realized using the IEC operands GT and SEL. If the measured temperature is higher than 55 °C the value 100 is output, otherwise a 0 results.

Depending on the used temperature sensor, the read value must first be normalized accordingly. In our example we assume a temperature sensor having a measuring range of 0 °C up to 100 °C and providing the measured temperature as 8 bit value in the form of a BYTE (value 0 = 0 °C, value 255 = 100 °C).

So the resulting program is as follows:

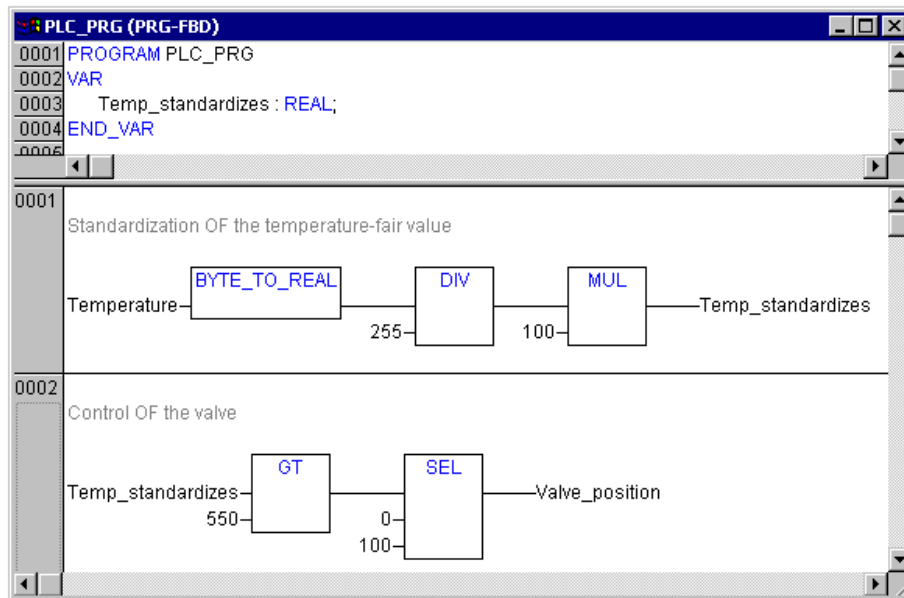


Figure 1-16 Program in 907 AC 1131

In addition, the condition of the coupler can be read out with the help of the function block DNM_STAT. If necessary, the detailed diagnostic information of the slaves can be polled using function blocks of the type DNM_SLVDIAG and an overview of the conditions of all slaves in the system can be polled using the function block DNM_SYSDIAG.

1.7 Diagnosis

1.7.1 Status LEDs

LED	Color	Status	Meaning
RDY	Yellow	On	coupler ready
		flashes cyclic	bootstrap loader active
		flashes irregularly	hardware or system error
		Off	defective hardware
RUN	Green	On	communication is running
		flashes cyclic	communication stopped
		flashes irregularly	missing or erroneous configuration
		Off	no communication
NET	Green/red	green on	connected to the bus, communication established
		flashes green	connected to the bus, no communication
		Off	no supply voltage, not connected to the bus
		red on	critical connection error
		flashes red	timing supervision error
MOD	Green/red	green on	coupler running
		flashes green	coupler ready for operation
		Off	no supply voltage
		red on	uncorrectable error
		flashes red	minor error

Table 1-4 Status LEDs of the DeviceNet coupler

1.7.2 DeviceNet Error messages

The DeviceNet error messages are listed in section 'Error messages of the internal coupler'.

1.7.3 Function blocks

DeviceNet master

- DNM_DEVDIAG – Reading the detailed diagnosis of a slave
- DNM_STAT – Reading the DeviceNet coupler status
- DNM_SYSDIAG – Displaying status information of all slaves

Online diagnosis

Ref. to documentation 907 FB 1131

1.8 Further information

1.8.1 Standardization

BOSCH CAN Specification – Version 2.0, Part A

ISO 11898

ODVA DeviceNet Specification Release 2.0, Errata 1 & 2

1.8.2 Important addresses

Open DeviceNet Vendor Association (ODVA)
PMB 499
20423 State Road 7 #F6
Boca Raton, FL 33498-6797
U.S.A.

Telephone: (+1) 954 340-5412
Telefax: (+1) 954 340-5413
Internet: <http://www.odva.org>

DeviceNet Europe
c/o Teja Ulrich
Elektrastraße 14
D-81925 München

Telephone: (+49) 8991049571
Telefax: (+49) 8991049573
e-mail: Teja.Ulrich@munich.netsurf.de

1.8.3 Terms, definitions and abbreviations used

ACK	Acknowledged Exchange
ODVA	Open DeviceNet Vendor Association
CAN	Controller Area Network
COS	Change Of State
EDS	Electronic Data Sheet
MAC ID	Media Access Control Identifier, bus address
Trunk Line	Main Strand of the bus
Drop Line	Branch line
UCMM	Unconnected Message Manager

2 Figures

Figure 1-1 Communication connections	1-2
Figure 1-2 Bus termination resistors	1-9
Figure 1-3 Example of a single master system.....	1-12
Figure 1-4 Example of a multi master system.....	1-13
Figure 1-5 Planning example	1-15
Figure 1-6 Inserting a master	1-16
Figure 1-7 Inserting a device.....	1-17
Figure 1-8 Bus overview	1-17
Figure 1-9 Device Configuration window.....	1-18
Figure 1-10 Configuration of the temperature sensor	1-19
Figure 1-11 Valve configuration	1-19
Figure 1-12 Gateway configuration	1-20
Figure 1-13 Downloading the configuration	1-21
Figure 1-14 Address table	1-21
Figure 1-15 Operands.....	1-22
Figure 1-16 Program in 907 AC 1131.....	1-23

3 Tables

Table 1-1 Structure of the Identity Object.....	1-5
Table 1-2 Assignment	1-8
Table 1-3 Maximum line lengths	1-11
Table 1-4 Status LEDs of the DeviceNet coupler	1-24

Content

1	Interbus master coupler	D 1-1
1.1	Brief overview	1-1
1.1.1	Fundamental properties and fields of application	1-1
1.1.2	Features	1-7
1.2	Technical data	1-9
1.2.1	Technical data of the coupler	1-9
1.2.2	Interface specification	1-9
1.3	Connection and transfer media	1-10
1.3.1	Attachment plug for the bus cable	1-10
1.3.2	Connection	1-11
1.3.3	Bus cable	1-13
1.3.4	Maximum line lengths	1-14
1.3.5	Optical fibers / converters	1-14
1.4	Possibilities for networking	1-15
1.4.1	Remote bus system	1-15
1.4.2	Multi segment system	1-16
1.5	Interbus - Implementation	1-17
1.5.1	Configuration	1-17
1.5.2	Running operation	1-17
1.5.3	Error diagnostics	1-17
1.5.4	Function blocks	1-18
1.6	Planning example	1-19
1.7	Diagnosis	1-32
1.7.1	Status LEDs	1-32
1.7.2	Interbus error messages	1-32
1.7.3	Function blocks	1-32
1.7.4	Online diagnostics	1-32
1.8	Further information	1-33
1.8.1	Standardization	1-33
1.8.2	Important addresses	1-33
1.8.3	Terms, definitions and abbreviations used	1-33
2	Figures	2-1
3	Tables	3-1

1 Interbus master coupler

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

Interbus was designed especially for the usage with machine systems and fast processes. Interbus is mainly used to transfer process data between a central controller module (such as PLC or PC) and decentralized peripheral devices such as I/O modules, sensors and actuators.

Ring structure

Interbus is a single master system. An essential difference between the Interbus and other often used field bus systems is the Interbus ring structure. The bus is set up as an active coupled ring, based on point to point connections. Each bus subscriber uses a separate two wire line for each transmission direction (forwards and backwards). The forward and return lines are combined in one cable. Due to this, the Interbus can be compared to a line bus, because the subscribers are only connected via one cable. This requires at least two physical interfaces at each slave in order to set up the ring structure.

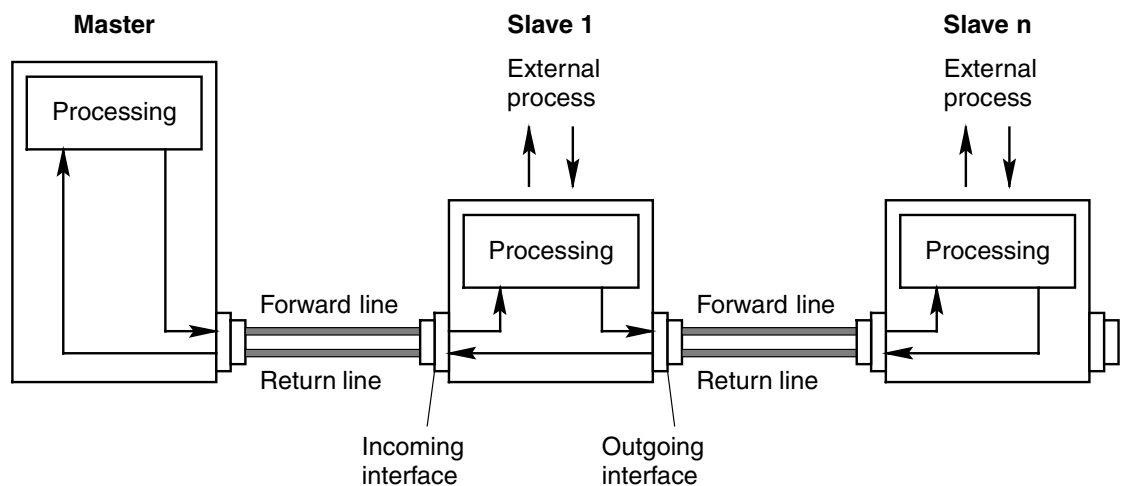


Figure 1-1: Realization of the ring structure using physical point to point connections

The transmitting and receiving interfaces are implemented as independent interfaces for each bus connection. This allows sending and receiving data simultaneously within one bus access (full duplex) and reduces transmission times. The backwards signal (sent to the master) is electrically prepared by each subscriber (active feedback). Thus, the slaves act additionally as repeaters. Due to this method transmission transfer lines up to 400 m length between two subscribers can be realized (remote bus). Assuming a maximum number of 512 subscribers (stations) and the usage of copper wires, the system can be extended over a theoretical length of 102 km. In practice an overall length of 12.8 km should not be exceeded. The data transmission rate is 500 kbit/s.

Interbus distinguishes different subscriber types and bus versions:

Subscriber types:

- Controller boards
- Bus terminals
- Remote bus devices / remote bus slaves
- Local bus devices / local bus slaves / peripheral bus slaves

Bus versions:

- Remote bus
- Installation remote bus
- Local bus / peripheral bus

The master of an Interbus system is referred to as a controller board. The Interbus main line connected to the master is called remote bus. If this remote bus includes the lines for data transmission and additionally the power supply for the connected I/O modules and the active sensors, it is referred to as the installation remote bus.

The so-called remote bus slaves and bus terminals are connected to the remote bus. While remote bus slaves (compact devices) have their own inputs or outputs respectively, the bus terminals are only used for the segmentation of the overall system. Typical bus terminals do not have own inputs or outputs. In addition mixed devices are obtainable complying both types of functionality.

Bus terminals differ from all other subscriber types because they have two outgoing interfaces. In addition bus terminals have their own power supply which is additionally used as supply for the connected I/O modules. The segment branching from a bus terminal can be carried out either as (installation) remote bus or local bus. Further bus terminals and remote bus slaves can be connected to a remote bus. This is referred to as the remote bus branch. So-called local bus devices are connected to a local or peripheral bus.

Local bus devices are I/O subscribers used for building up a non-central substation at which a great number of signals have to be connected within a short gap. A typical operating site is the control cabinet. The subscribers are coupled to the remote bus via a bus terminal. In addition they are supplied from the bus terminal. Within the local bus, no branches are allowed.

The maximum total expansion of a segment, the maximum number of subscribers within one segment as well as the distance between them depend on the bus configuration.

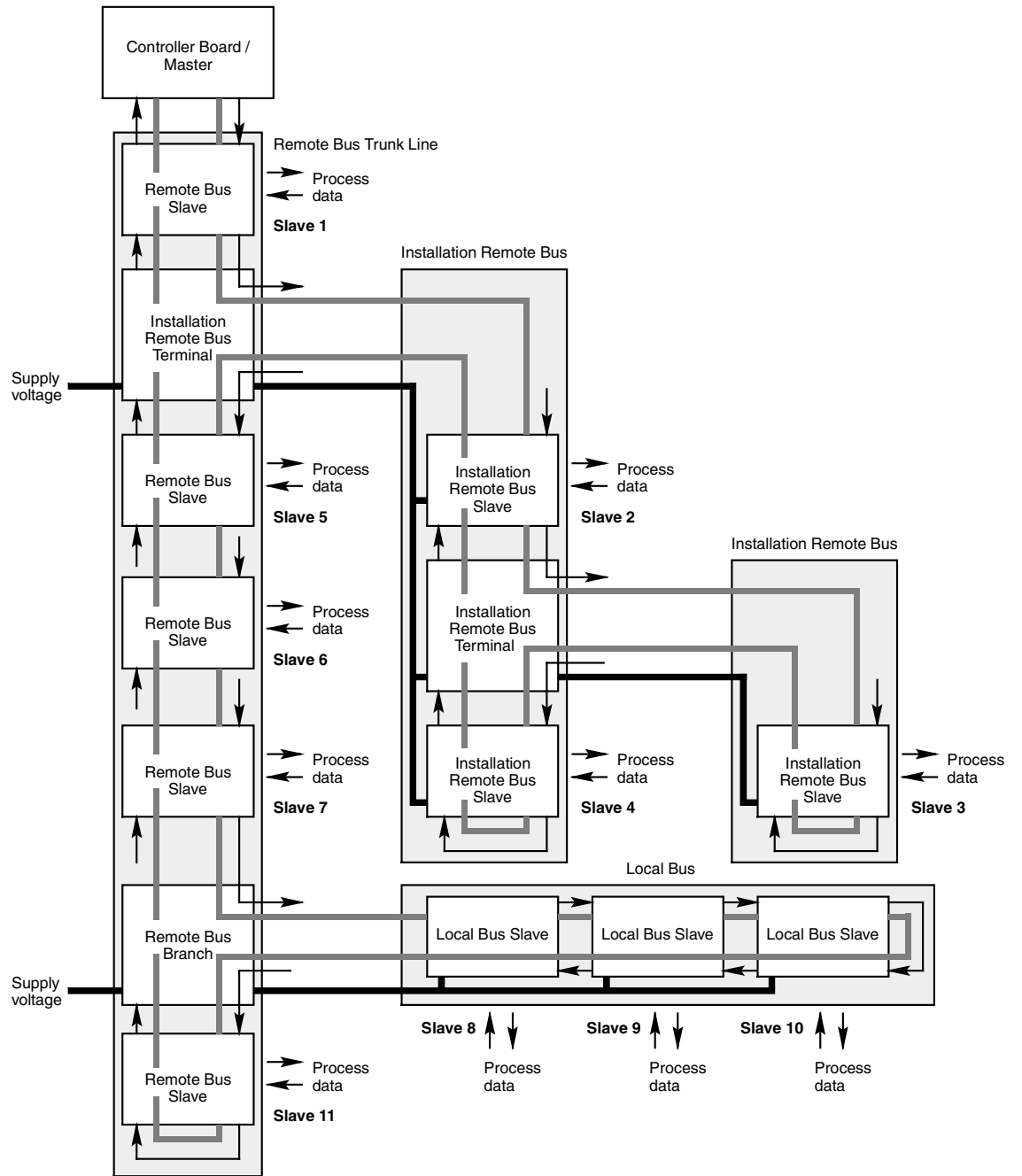


Figure 1-2: Topology

Segmentation

Using control codes the outgoing interfaces of the remote bus slaves and the bus terminals can be connected or disconnected (e.g. in case of an error). Here it has to be observed that a local bus can only be set into/out of operation as an entire segment via a bus terminal because local bus modules itself have no interface switching function. If an outgoing interface in the (installation) remote bus is switched off, all further connected interfaces are switched off too.

The counting order of the subscribers starts at the slave which is directly connected to the master. The branching interface of a bus terminal is physically located before the outgoing interface of the higher-level line. This means that the numbering of the subscribers is first continued in a segment looped in via a bus terminal before returning to the segment with the higher order.

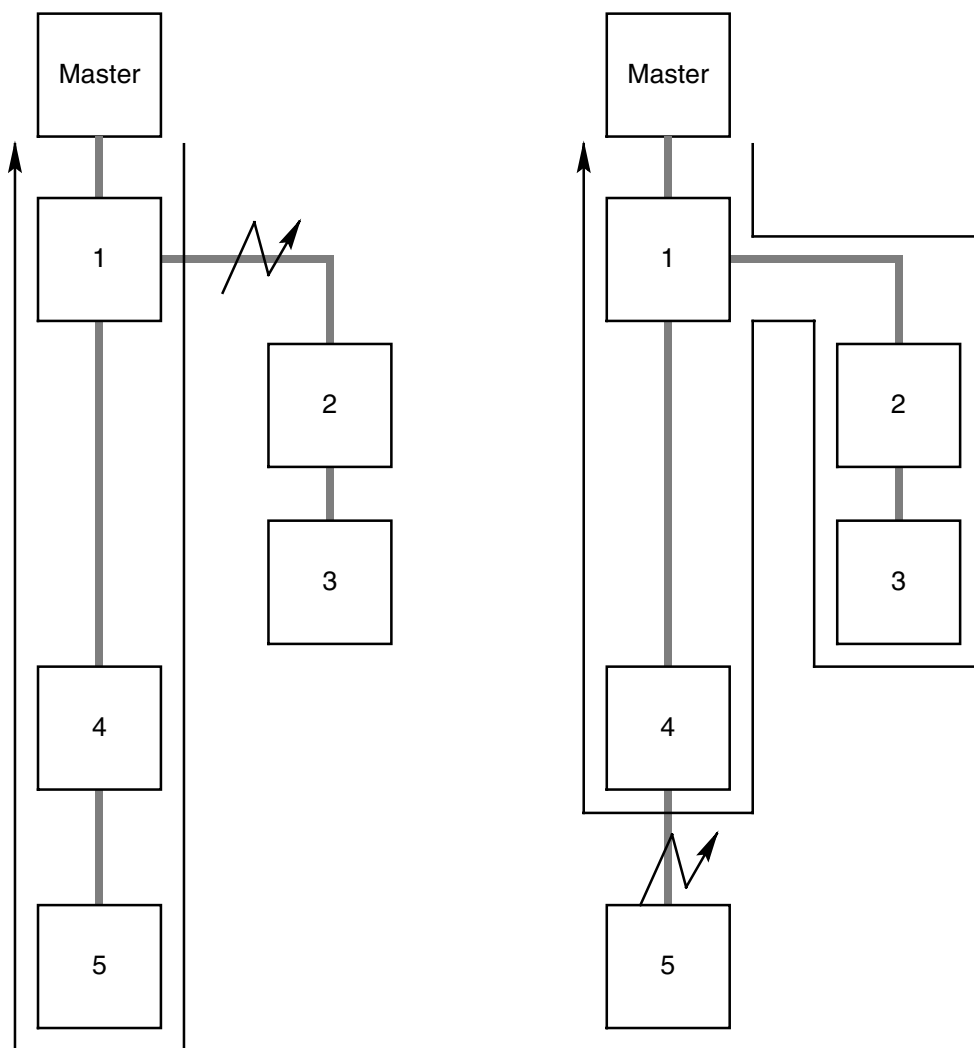


Figure 1-3: Segmentation

Commissioning

Prior to the actual process data exchange between master and slaves, the Interbus system must be connected by the master. For that purpose the master reads the ID code and the size code of each subscriber during several identification cycles by connecting and disconnecting the various bus segments. The connecting process is completed, if no new subscriber is detected within two identification cycles. After the connecting process a comparison is performed between the detected actual configuration and the predetermined reference configuration. The cyclic data exchange only starts if both data sets match.

Protocol

The Interbus protocol mechanism bases on a so-called summation frame protocol. In conjunction with the used ring topology the data are shifted as through a shift register. All output data destined for the slave subscribers are contained in the telegram sent by the master. Each slave reads the output data out of the telegram which are destined for it. The input data of the appropriate slaves destined for the master are inserted into the telegram by the slave at the corresponding position.

In Interbus it is not necessary to set slave addresses, because the addresses are determined by the subscriber position in the ring structure.

Parameter data channel (PCP)

Some slave subscribers provide an additional parameter data channel (PCP, Peripherals Communication Protocol) beneath the cyclic process data exchange. Parameter data which have to be inserted into the summation frame telegram are not time-critical. To avoid an unnecessary extension of the summation frame telegram, this expansion usually includes not more than 2 bytes for each slave which is able to communicate. If a parameter set exceeds this size it is split in several data packets and transmitted sequentially. The receiver recombines the single packets after the complete transmission in order to get the total data set. Parameter data can be sent from the master to a slave as well as from a slave to the master.

This kind of non-cyclic communication is used for example when controlling drives via the Interbus (see Interbus profiles 21 and 22, DRIVECOM). Via the PCP channel the reference rotational speed can be preset or modified (among other values).

Status telegrams

The master additionally sends status telegrams between the transmission of the telgrams of the identification cycle and the data cycle as well as in cycle pauses. These status telegrams do not contain user data and they are passed from each subscriber to the subsequent subscriber without any modifications. They are only used to indicate the master activity to the subscribers and to maintain communication. If a slave recognizes the connection to the master due to an incoming status telegram it activates the green RC LED (Remote bus Check). This allows to verify the physical connection between a subscriber and the master in a simple way prior to commissioning the network.

Transmission reliability

For detecting errors within a transmission cycle, the Interbus protocol uses the Cyclic Redundancy Check (CRC). The checksum generated when applying this method is sent by the master after each cycle, passed from slave to slave and verified for consistency by each slave. If a slave determines a corrupt checksum, it ignores the previously received output data. Via the Interbus ring structure the error information sent by the slave is retransmitted to the master which initiates a new transmission cycle with the current output data. By applying this method Interbus achieves a Hamming distance (measure of data transmission reliability) of 4.

Diagnostics

Interbus subscribers usually have several LEDs indicating the device status on the site. In addition slave devices report the following error status to the master:

- CRC error
- Module error
- Reconfiguration error

These information are contained in the identification codes of the modules. Using the CRC error bit each slave reports, whether it has detected a CRC error during the last identification cycle or data cycle. By evaluating the CRC error bits of all slaves the master is able to exactly localize a possible fault position in the bus.

If a slave reports a module error, the device has detected a peripheral power supply failure and/or a short circuit in at least one of the local peripheral inputs or outputs. A module error only affects the process inputs of the concerned slave. The bus operation is not affected.

Bus terminals often provide a reconfiguration button which is accessible from the outside. Pressing this button on the site (e.g. because an additional slave module was installed) this is reported to the master using the reconfiguration error bit. As a result the master should be reconfigured in order to enable it to detect the planned system expansion.

Additionally to this standard diagnosis possibilities, the Interbus master coupler detects many other error status.

1.1.2 Features

Mode of operation:

- Interbus master, controller board
- Automatic identification of all bus subscribers
- Checking the reference configuration and actual configuration when initializing the system
- Up to 128 slave subscribers in all
- Up to 12 bus segment levels
- A maximum of 4096 I/O points

Transmission technique:

- RS422 according to DIN 19258
- Data transmission rate 500 kbit/s
- Interfaces Remote out (floating), Remote in (non-floating)
- (Remote) bus connector 9-pin, SUB-D socket
- Data lines (remote bus): 6 cores, twisted pair, common shielding
- Transmitting and receiving lines combined in one cable. No additional line is necessary for closing the ring.

Communication:

- Bus access master-slave using the summation frame telegram
- Cyclic I/O data exchange
- PCP parameter data channel
 - Supported services: Read, Write, Identify
 - Up to 62 communication references
- Loop support
- Generation 4 support

Protection functions:

- Data transfer with Hamming distance $HD = 6$.
- Avoiding of erroneous configurations, because the reference configuration is compared to the actual detected configuration when starting up the system and the user data operation is only started if both configurations match.
- Operation monitoring of all subscribers using status telegrams.
- Integrated diagnostic functions.
- Configurable behavior of the master in case of an error:
 - In case of a subscriber failure or segment failure the system keeps running. The failure is registered in the master and indicated via a common diagnosis. In this mode removing, adding and exchanging of subscribers during running operation is possible.
 - Communication is stopped if a subscriber is missing.
 - Communication is stopped if a subscriber reports a peripheral error.
 - Communication is stopped if a subscriber is missing or reports a peripheral error.

Status indication via 3 LEDs:

- READY (yellow): The coupler is ready for operation.
- RUN (green): Communication status
- STA (yellow): not used
- ERR (red): Interbus error

1.2 Technical data

1.2.1 Technical data of the coupler

Coupler type	Interbus master coupler in PC/104 format
Processor	16 bit processor with interrupt controller and DMA controller
Memory expansion	2 KBytes DP-RAM, 512 KBytes Flash EPROM, 128 KBytes RAM
Internal supply	+5 V, 650 mA
Dimensions	96 x 90 x 23 mm
CE sign	55011 Class B for Emissions, EN 50082-2 for Immunity

1.2.2 Interface specification

Interface socket	9-pin, SUB-D socket
Transmission standard	RS422 according to DIN 19258, Remote out (floating), Remote in (non-floating)
Transmission protocol	Interbus, 500 kBaud max.
Transmission rate	500 kBit/s
Status indication	using 3 LEDs
Number of slaves	128 max.
Number of bus segment levels	12 max.

1.3 Connection and transfer media

1.3.1 Attachment plug for the bus cable

9-pole SUB-D connector

Assignment:

Pin No.	Typical core color	Signal	Meaning
1	Green	DO2	Transmitting data line +
2	Pink	DI2	Receiving data line +
3	Brown	COM	Equalizing line
4			
5	-	Udd	Logic voltage 5 V
6	yellow	/DO2	Transmitting data line -
7	grey	/DI2	Receiving data line -
8			
9	-	BCI	Activation of the bus connector

Table 1-1: Assignment

It is absolutely necessary to connect all lines. The transmitting data lines DO2 and /DO2, the receiving data lines DI2 and /DI2 and the equalizing line GND2 are connected to the next subscriber via the bus connector. The signals Udd and BCI have to be bridged directly at the bus connector (9-pin SUB-D plug) in order to activate the interface.

Supplier:

e.g. SUBCON 9/M-SH

Phoenix Contact GmbH & Co.

Flachmarktstraße 8 - 28

D-32825 Blomberg

Telephone: (+49) (0)52 35 / 3-00

Telefax: (+49) (0)52 35 / 3-4 12 00

Internet: <http://www.phoenixcontact.com>

1.3.2 Connection

Due to the Interbus' active ring topology no termination resistors are necessary at the bus endings. The following figure illustrates the connection of the first slave to the master for different common bus connectors on the slave side.

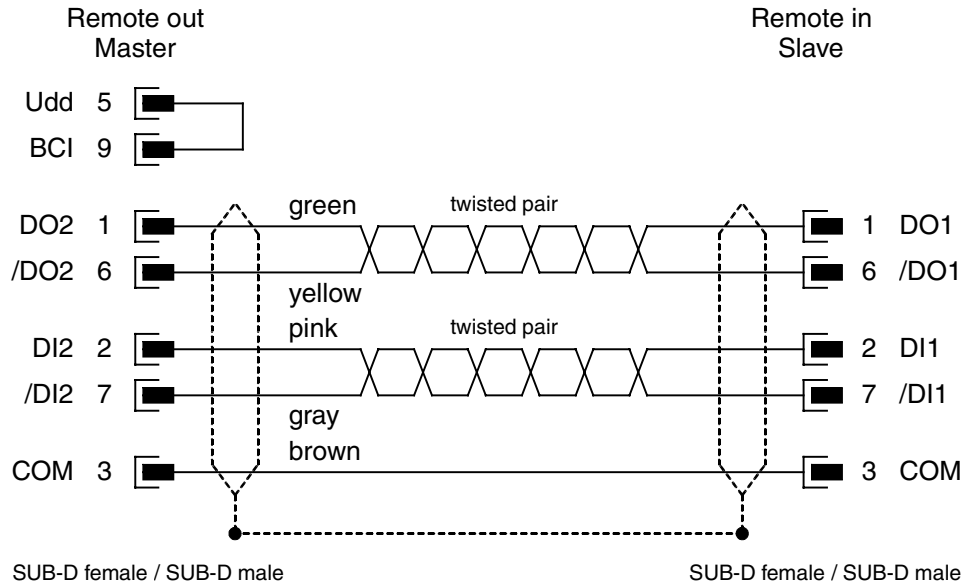


Figure 1-4: Connection SUB-D 9-pin to SUB-D 9-pin

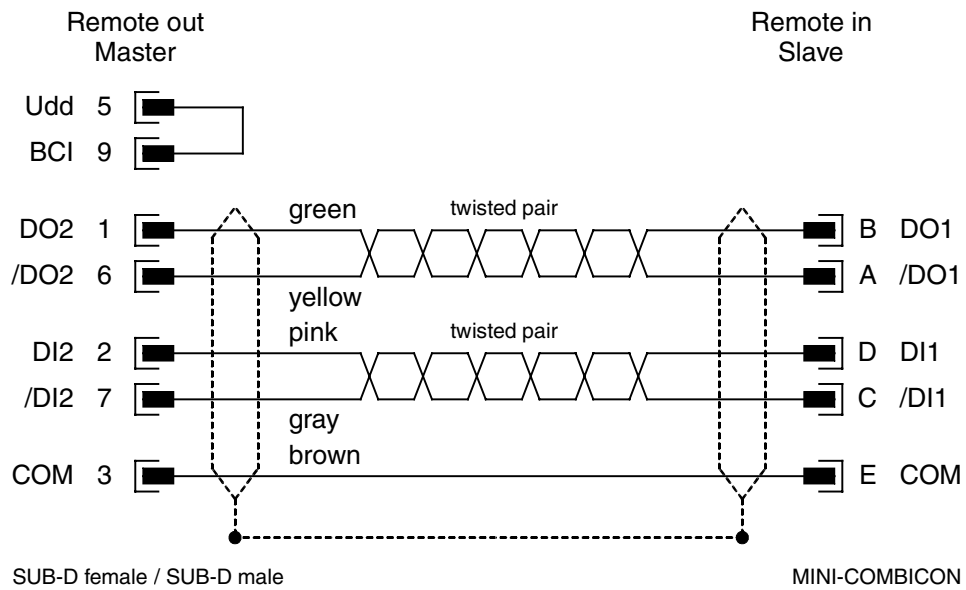


Figure 1-5: SUB-D 9-pin to MINI COMBICON 5-pin or 10-pin

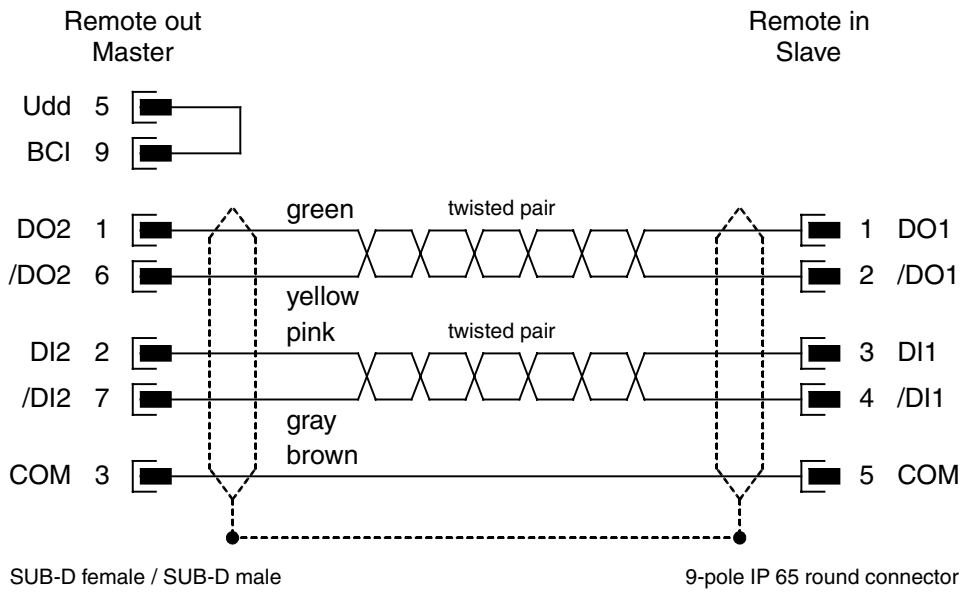


Figure 1-6: SUB-D 9-pin to IP65 circular connector 9-pin

1.3.3 Bus cable

In Interbus different bus cables are used, depending on the bus segment configuration. Whereas remote bus cables only have data lines, installation remote bus cables and local bus cables also include additional cores for the power supply of connected devices. In addition, local bus cables vary depending on the required class of protection. Due to the different application-specific configurations of the local bus cables, this documentation only describes the two remote bus cable types. When using local bus devices, information about the required cables can be found in the related device documentation.

Remote bus cable

Type:	3 x 2 cores, twisted pair, Color encoding according to DIN 47100 common shielding made of tinned copper netting
Data lines	
Wave impedance (cable impedance):	100 Ω
Cable capacity (distributed capacitance):	≤ 60 nF/km
Line cross section:	≥ 0.22 mm ²
Line resistance per core:	≤ 93 Ω /km
Loop resistance (resistance of 2 cores):	≤ 186 Ω /km

Installation remote bus cable

Type:	3 x 2 cores, twisted pair, Color encoding according to DIN 47100 (data lines); 3 single cores red, blue, green/yellow (power supply); common shielding made of tinned copper netting
Data lines	
Characteristic impedance:	100 Ω
Cable capacity (distributed capacitance):	≤ 60 nF/km
Line cross section:	≥ 0.25 mm ²
Line resistance per core:	≤ 93 Ω /km
Loop resistance (resistance of 2 cores):	≤ 186 Ω /km
Power supply	
Line cross section:	≥ 1.0 mm ²

Supplier:

e.g. UNITRONIC® BUS IBS
U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
D-70565 Stuttgart
Telephone: (+49) (0)711 7838 01
Telefax: (+49) (0)711 7838 264
Internet: <http://www.lappkabel.de>

1.3.4 Maximum line lengths

The maximum total expansion of a segment, the maximum number of subscribers within one segment as well as the distance between them depend on the segment configuration, i.e. whether it is used as remote bus, installation remote bus or local bus.

Segment type	Remote bus	Installation remote bus	Local bus
Max. number of subscribers	254	254	8
Max. current consumption	-	4.5 A	0.8 A
Maximum line lengths (copper lines):	Between master and first remote bus subscriber 400 m	Between bus terminal and first I/O subscriber 50 m	Between bus terminal and first local bus module 1.5 m
	Between two remote bus subscribers 400 m	Between two I/O subscribers 50 m	Between two local bus modules 1.5 m
	Between master and last remote bus subscriber 12.8 km	Between bus terminal and last I/O subscriber 50 m	Between bus terminal and last local bus module 10 m

Table 1-2: Maximum line lengths

1.3.5 Optical fibers / converters

For applications in areas with strong electromagnetic interference sources (EMC), Interbus uses optical fibers in order to prevent from possible negative influences on the data transmission. For that purpose interface converters are used performing electrical/optical and optical/electrical conversion.

Supplier:

e.g. EPIC® BUS

CONTACT GmbH
 Gewerbestraße 30
 D-70565 Stuttgart
 Telephone: (+49) (0)711 7838 03
 Telefax: (+49) (0)711 7838 366
 E-mail: contact@contactconnectors.de

1.4 Possibilities for networking

The Interbus master coupler is connected to the bus via the 5-pin SUB-D socket. As an EMC measure and for protection against dangerous contact voltages, the shield of the bus line must be connected to protective earth outside the housing.

1.4.1 Remote bus system

The simplest configuration of the Interbus system is a pure remote bus line. It consists of a control with an Interbus master coupler used as master and up to 128 I/O modules operating as remote bus slaves.

The Interbus master is able to:

- automatically determine the structure of the system, i.e. the structure and segmentation of the entire Interbus system and its subscribers as well as the I/O configuration of each subscriber
- compare the actual detected system with the configuration data and therefore to verify the configuration
- start and stop communication
- read input data of the slaves
- write output data of the slaves
- read and write parameter data of slaves supporting PCP
- monitor the availability of the slaves
- read diagnostic data of the slaves

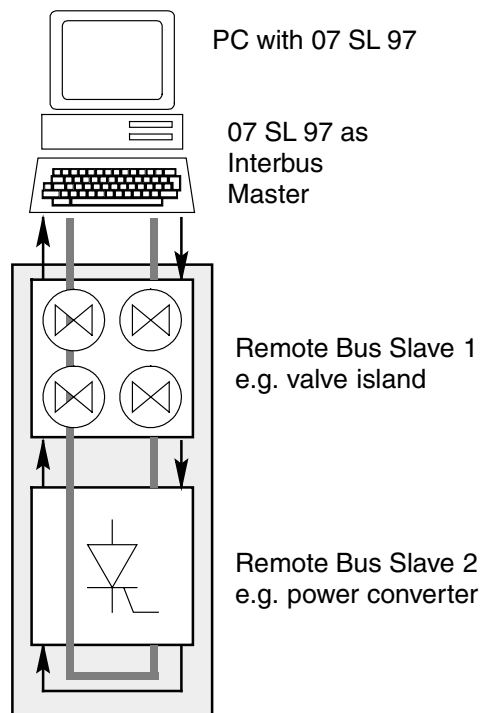


Figure 1-7: Example of a simple remote bus system

1.4.2 Multi segment system

An Interbus system can be segmented using bus terminals. The configuration of a segment as remote bus, installation remote bus or local bus is application-specific and is determined by the type of the bus terminal. Within one bus segment only subscribers can be connected, which can be operated with the corresponding bus type. A maximum of 128 slaves can be used in the system.

The Interbus master is able to:

- automatically determine the structure of the system, i.e. the structure and segmentation of the entire Interbus system and its subscribers as well as the I/O configuration of each subscriber
- compare the actual detected system with the configuration data and therefore to verify the configuration
- start and stop communication
- read input data of the slaves
- write output data of the slaves
- read and write parameter data of slaves supporting PCP
- monitor the availability of the slaves
- read diagnostic data of the slaves

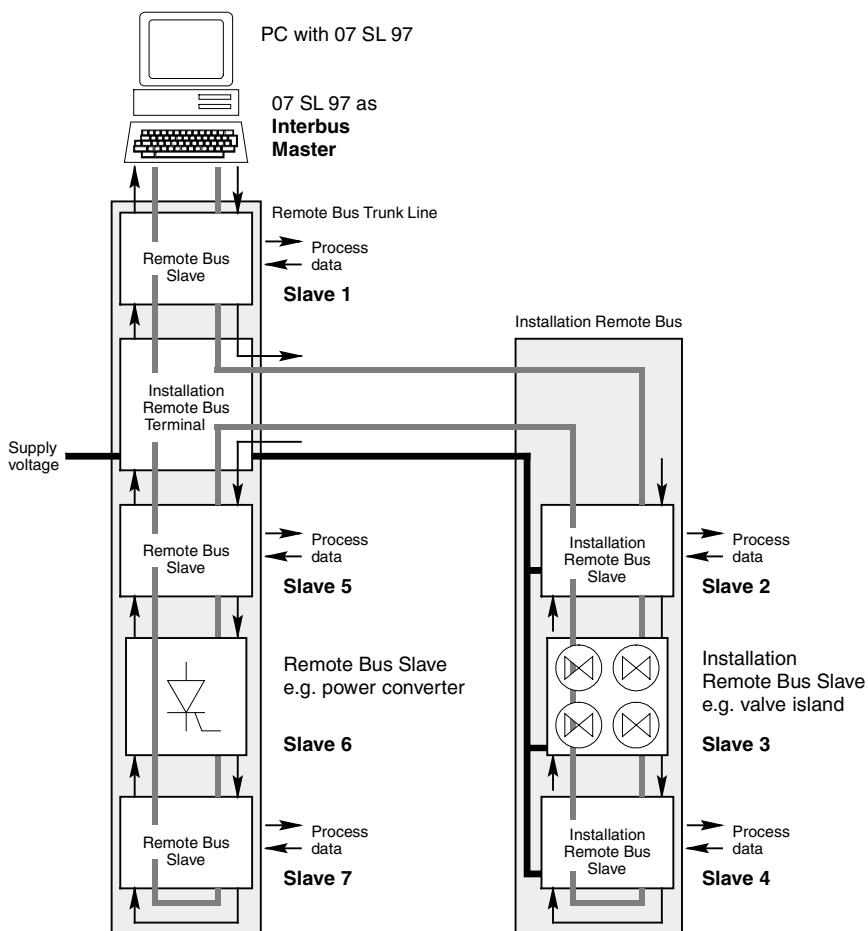


Figure 1-8: Example of a multi segment system

1.5 Interbus - Implementation

1.5.1 Configuration

The Interbus master coupler is configured via the PC using the 907 FB 1131 configuration software for field bus couplers (see documentation 907 FB 1131 and configuration example). Configuration data created using 907 FB 1131 are not assigned directly to a project. Thus, the resulting file has to be downloaded separately to the control (additionally to a 907 AC 1131 project). It is there stored in a Flash memory automatically.

If the user defined project is written to SMC, also the configuration data are automatically saved.

1.5.2 Running operation

The Interbus protocol is automatically processed by the coupler and the operating system of the control. The coupler is only active on the bus if it was correctly initialized before and if the user program runs. No function blocks are necessary for exchanging process data via Interbus. Special Interbus functions can be realized using the 907 AC 1131 function blocks of the Interbus library (see documentation of library Interbus_Master_S90_Vxx.LIB + Coupler_S90_Vxx.LIB).

When starting the user defined program, the coupler starts the communication via Interbus and first compares the stored reference configuration with the actual detected bus configuration (segmentation, number and types of subscribers, I/O configuration). If all information match, the process data exchange with the slaves is started automatically.

If the user defined program is stopped, the coupler shuts down the Interbus communication in a controlled manner.

1.5.3 Error diagnostics

Interbus communication errors are generally indicated by the LEDs beside the bus connector. A malfunction of the Interbus driver in the control or the coupler itself is indicated via the FK error flags and the corresponding LEDs (see to error tables). Furthermore the Interbus library provides various function blocks which allow a detailed error diagnosis (see documentation of library Interbus_Master_S90_Vxx.LIB + Coupler_S90_Vxx.LIB).

1.5.4 Function blocks

Interbus_Master_S90_V43.LIB + Coupler_S90_V41.LIB

General

- INTERBUS_INFO: Reading of coupler information

Interbus master

Status / Diagnostics

- IBM_STAT: Reading the coupler status
- IBM_SLVDIAG: Reading the detailed diagnosis of a slave
- IBM_SYSDIAG: Reading the system diagnostics PCP services
- IBM_GET_OD: Reading object descriptions from a slave supporting PCP
- IBM_READ: Writing object descriptions to a slave supporting PCP
- IBM_READ_EN: Enabling the reading of parameters from the control via a slave supporting PCP
- IBM_WRITE: Writing parameters to a slave supporting PCP
- IBM_WRITE_EN: Enabling the writing of parameters to the control via a slave supporting PCP

1.6 Planning example

Below, a planning example for the 07 KT 97 R 169 used as Interbus master is shown. Beneath 07 KT 97 used as Interbus master, remote bus slaves are connected to the bus, one with an analog input and the other with digital outputs. A temperature sensor is connected to the analog module, a valve is connected to the digital output module. So the resulting configuration is as follows:

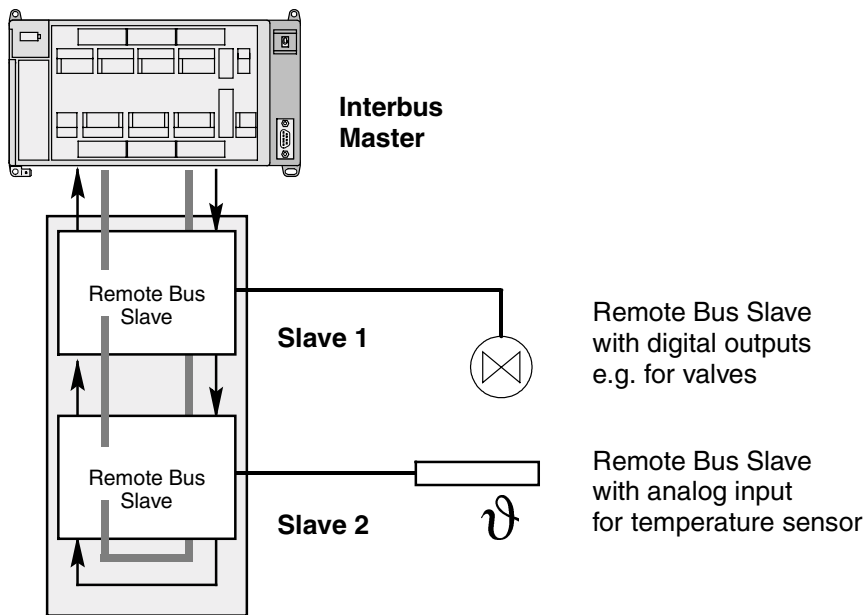


Figure 1-9: Planning example Interbus

The control used as Interbus master reads the temperature values measured by the sensor. Depending on this value, a control signal is generated and sent to the valve.

Configuration with 907 FB 1131

Due to the Interbus ring structure and the summation frame telegram used for communication, it is not necessary to set the slave bus addresses. The assignment of I/O data to a slave results from the slave position within the system.

Each Interbus slave device knows its own device information (ID code) and I/O configuration (size code). These information are first read out by the master from all connected slaves when starting up Interbus and then compared with the reference configuration predefined by the user. Alternatively the available actual system configuration can be read by the master in 907 FB 1131 and then be applied as reference configuration. This guarantees that the configuration data agree to the actual system description. However, this method assumes that the system is already installed completely and correctly. For that reason the following example describes how to configure the coupler in offline mode.

For configuring the Interbus master coupler first start the configuration software 907 FB 1131. In the menu *File* select *New*. A window appears listing all available field bus systems. In this window select *Interbus* and confirm with *OK*. Except from a rectangular marking frame the main window is now empty. Save the file under any name in any directory by selecting *File – Save as...* (e.g. *Interbus_Master_Sample*). The file name automatically gets the extension *.ib* (**InterBus**).

If the slave modules are used the first time (as described in this example), you first have to create the corresponding EDS file for each device (see 907 FB 1131 documentation). EDS files describe Interbus subscribers and contain information about their configuration (input/output, digital/analog, support of PCP, ID code) and additional information such as vendor name, device designation and description as well as references to graphical representations.

Three figures are required for the graphical representation of a device in 907 FB 1131: configuration, operation and diagnostics. For this, you can either use available default files or create individual representations for the appropriate device. If the corresponding figures are already available in the required format (16 colors bitmaps, 70 pixels wide, 40 pixels high) you can insert the figures directly by copying the files into the 907 FB 1131 subdirectory *Fieldbus|Interbus|Bmp*. Once the bitmap files are created they can be used several times, i.e. a figure representing a valve can be used for all valve descriptions. If you want to use already existing bitmaps in this example, skip the next section and proceed with the section

Creating individual symbols for the graphical device representation

All bitmap files used for the graphical representation of Interbus subscribers are located in the 907 FB 1131 subdirectory *Fieldbus|Interbus|Bmp*. The bitmaps must have the following properties: 16 colors, width 70 pixels, height 40 pixels. New bitmap files can be created in two various ways. You can either modify an already existing figure and save it under a new name or create a new representation.

To edit an existing figure, start the Explorer and browse to the 907 FB 1131 subdirectory *Fieldbus|Interbus|Bmp*. Start your default graphic tool (e.g. Paint) by double clicking on one of the listed bitmap file icons. It is also possible to start the graphic tool directly and to open the desired bitmap file afterwards.

In order to draw a new bitmap, start your graphic tool (e.g. Paint) and then create a new file via the tool menu (e.g. *File – New*).

Now the bitmap properties have to be set. When using Paint please select *Image – Attributes...* and enter the values width 70 pixels, height 40 pixels and select the colors. Save this (empty) file as 16 colors bitmap under any name (e.g. blank.bmp) into the 907 FB 1131 subdirectory *Fieldbus|Interbus|Bmp*. This file can be used as template for future bitmaps.

Now it is recommended to create the figure which represents the device in configuration mode. This representation usually contains a normal figure of the device in front view or in perspective view. Start with the configuration representation of the valve used in our example by drawing the appropriate figure in neutral colors. Save the bitmap as *Valve_DEF.bmp* in the 907 FB 1131 subdirectory *Fieldbus|Interbus|Bmp*. Now draw the representation of the valve in running mode and save it as *Valve_Run.bmp*. This figure represents the normal valve operation (e.g. green). Now proceed as described for the diagnostics representation (*Valve_DIAG.bmp*), which is to be displayed in the 907 FB 1131 Debug mode if an error was reported by the valve (e.g. red).



Ventil_def.bmp



Ventil_run.bmp



Ventil_diag.bmp

Figure 1-10: Examples for the graphical valve representation in configuration, running and diagnostics mode

Create the corresponding symbols for the temperature sensor and save them as Temp_DEF.bmp, Temp_RUN.bmp and Temp_DIAG.bmp.



Temp_def.bmp



Temp_run.bmp



Temp_diag.bmp

Figure 1-11: Examples for the graphical temperature sensor representation in configuration, running and diagnostics mode

Creating EDS files

Each Interbus device is described using an ID code and a size code. If the Interbus master coupler is to be configured in offline mode (i.e. without reading the actual system configuration via the bus), an EDS (Electronic Data Sheet) file is required for each subscriber type containing the properties of the related device. Several EDS files are already delivered with 907 FB 1131. If further device types are to be used, the corresponding EDS file has to be created prior to the initial usage. 907 FB 1131 provides a comfortable user interface for creating these files. Once a device description was created using this editor, it can be reused for each device of this type, even if the devices are within the same configuration.

In this example the EDS files for the valve and the temperature sensor (both used as Interbus slave) have to be created. For that purpose, in 907 FB 1131 start the EDS file editor by selecting *Tools – EDS Generator* and enter the valve properties into the input dialog. Enter your name into the field *File – Created by*. The text entered into this field only serves as additional information (e.g. if several persons are creating EDS files); it has no functionality concerning the configuration. Fill in the fields *Device*, *Short description* and *Vendor* in the *Description* area. The data entered into these fields are used for managing and organizing the EDS files and facilitate a future selection of a desired device. Enter the correct device designation given by the vendor into the field *Device* (e.g. „SampleValve" in our example). The field *Short description* allows you to specify a short name for devices of this type (e.g. „valve"). Enter the name of the device manufacturer into the field *Vendor* (e.g. „Sample Inc.").

In the *Configuration* area below, the most important information are summarized. These entries must absolutely match with the actual device properties. Wrong entries result in configuration errors and the Interbus system can not start up when commissioning.

Since the used valve is a remote bus slave having its own process connection without bus terminal properties (segment branch), you have to select *Remote bus device* from the pull down menu *Type*. A valve is only operated as output module. For that reason select *Process data direction Output*. The used valve only provides the functionality Open/Close (no controlling valve) and does not support PCP. Due to this select *digital* for the *Device class* and *Output size 1 Bit* for the *Process data size*. Using these entries the *Size code* for the device is generated automatically. The *Device identification* list now contains all possible ID codes, depending on the previously set values. For the ID code to be selected please refer to the device description or to the device label. Select the ID code 1 for this example.

Finally you have to select, how the device should be represented in offline mode (during configuration phase) and online mode (while debugging in normal operation and in case of an error) in 907 FB 1131. In the *Bitmap*

area default symbols are already predefined for these three representations. You can either leave these bitmaps unchanged or replace the selected bitmaps by any other files. The bitmap files must have the properties 16 colors, width 70 pixels, height 40 pixels and have to be stored in the 907 FB 1131 subdirectory *Fieldbus\Interbus\Bmp*. You can display a preview of the currently selected figures by clicking on the *Layout* button.

If you have performed all settings, the input dialog should look as follows (assuming that you have inserted the previously created figures):

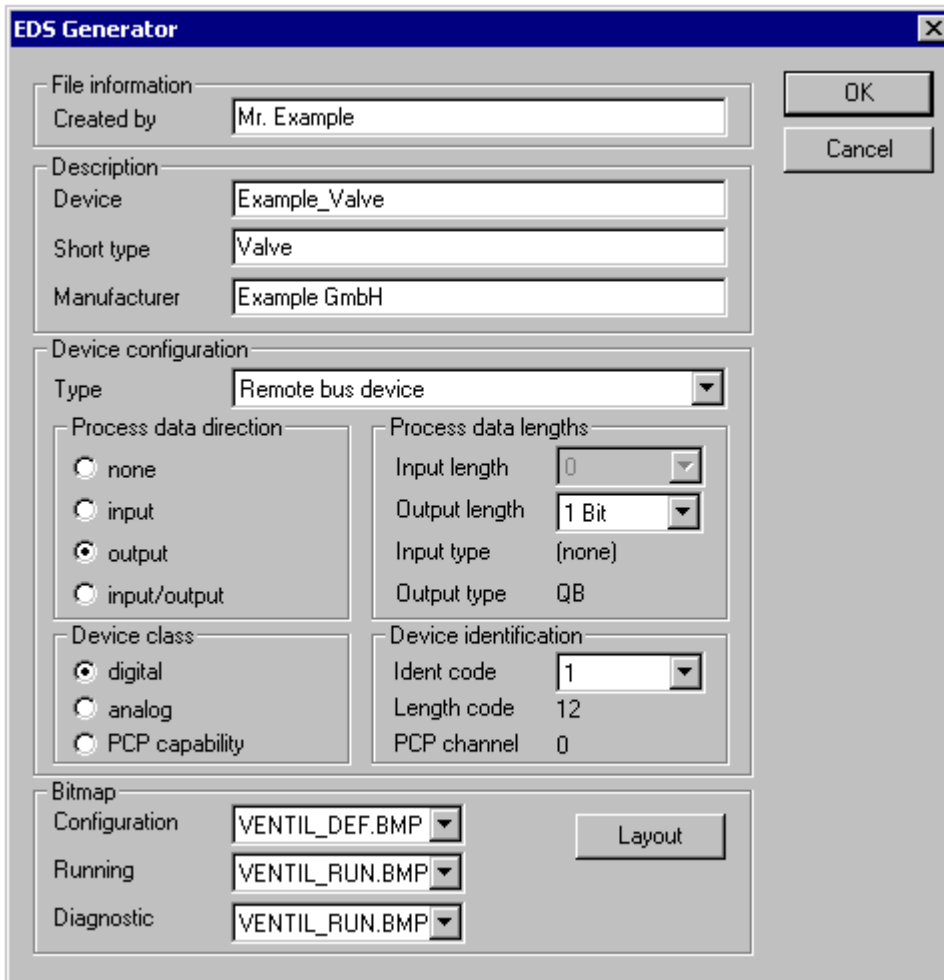


Figure 1-12: EDS_Editor_Ventil.bmp


Click on the *OK* button and save the new EDS file under any name using the extension *.eds* (e.g. *valve.eds*) in the predefined directory (907 FB 1131 subdirectory *Fieldbus|Interbus|Eds*). Afterwards repeat the procedure and create an EDS file for the temperature sensor using the following properties:

Device: SampleTemperatureSensor
 Short description: Temperature sensor
 Vendor: Sample Inc.
 Type: Remote bus device
 Process data direction: input
 Device class: analog
 Input size: 12 bit
 ID code: 50

The sensor is a remote bus input module providing the measured temperature as a 12 bit analog value via the Interbus. Save the new EDS file as *TempSens.eds*.

After this, the creation of the required EDS files is completed.

Creating the system configuration

Now insert the 07 KT 97 as Interbus master. For that purpose click on the button *Master* . Select the item *07 SL 97-IBM* in the *Available devices* list and click on the button *Add >>*. After this, the 07 KT 97 is inserted as Interbus master into the *Selected devices* list. Enter an unique device description (e.g. *SampleMaster*) and confirm with *OK*.

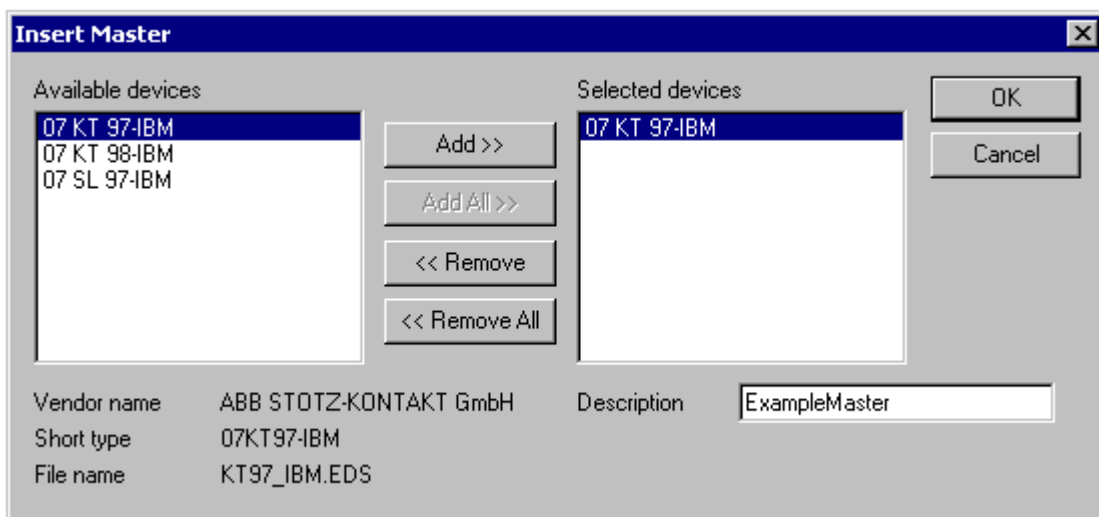

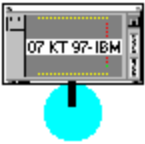


Figure 1-13: Inserting the master *Window.bmp*

Now insert the remote bus slaves in the same way. Select the button *Remote bus device*  and click into the turquoise colored area below the previously inserted master in the main window.

Each turquoise marked area displays all allowed positions within the Interbus system at which a device of the previously selected class can be inserted. Since only the master is already inserted, there is only one position for inserting the remote bus slave. In larger systems, segmented using bus terminals, a device can be inserted at various positions. After selecting the device class to be inserted, 907 FB 1131 marks all allowed positions within the system by a turquoise dot. Click onto the position at which the corresponding device is to be inserted.



Select *SampleValve* in the *Available devices* list and click on the button *Add >>*. If the desired entry is not visible, expand the list of displayed *Vendors* to *All* or to *Sample Inc.* When adding the valve, it is inserted into the *Selected devices* list as remote bus slave. Now enter an unique device description (e.g. valve). Change again to the list containing the available devices in order to select the second slave. Select the entry *Temperature sensor* and apply your selection with *Add>>*. Enter the description "Temperature sensor" for this slave. Confirm with *OK*.

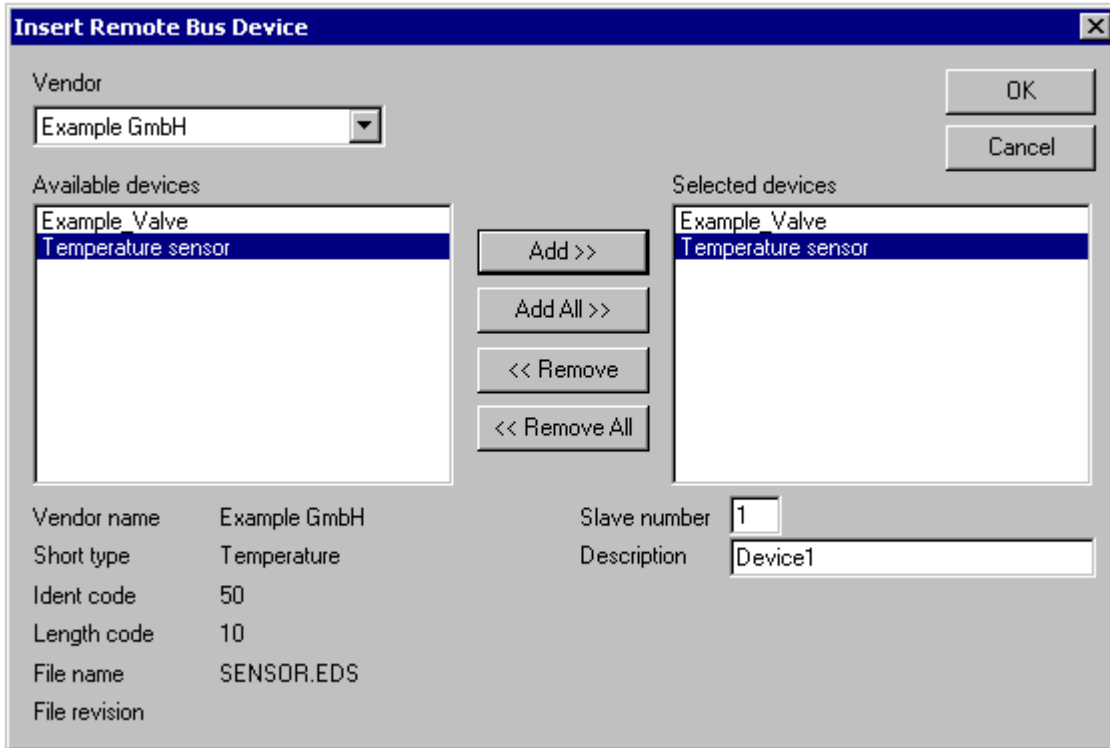


Figure 1-14: Inserting a remote bus device Window.bmp

The resulting bus view corresponds to the physical device arrangement.

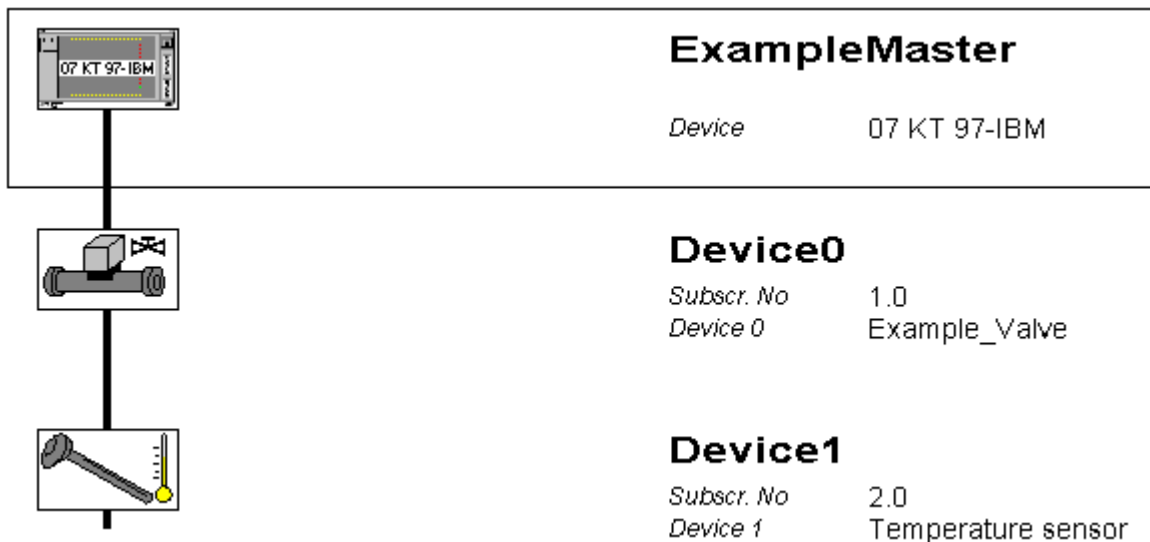


Figure 1-15: Bus_overview.bmp

In the previous steps the subscribers connected to the bus were described. The ID and size codes of the individual slave devices determine, which data are exchanged between each device and the master. Detailed information about the configuration of each slave device can be obtained by double clicking with the left mouse button on the device icon. In case of the temperature sensor the following window appears.

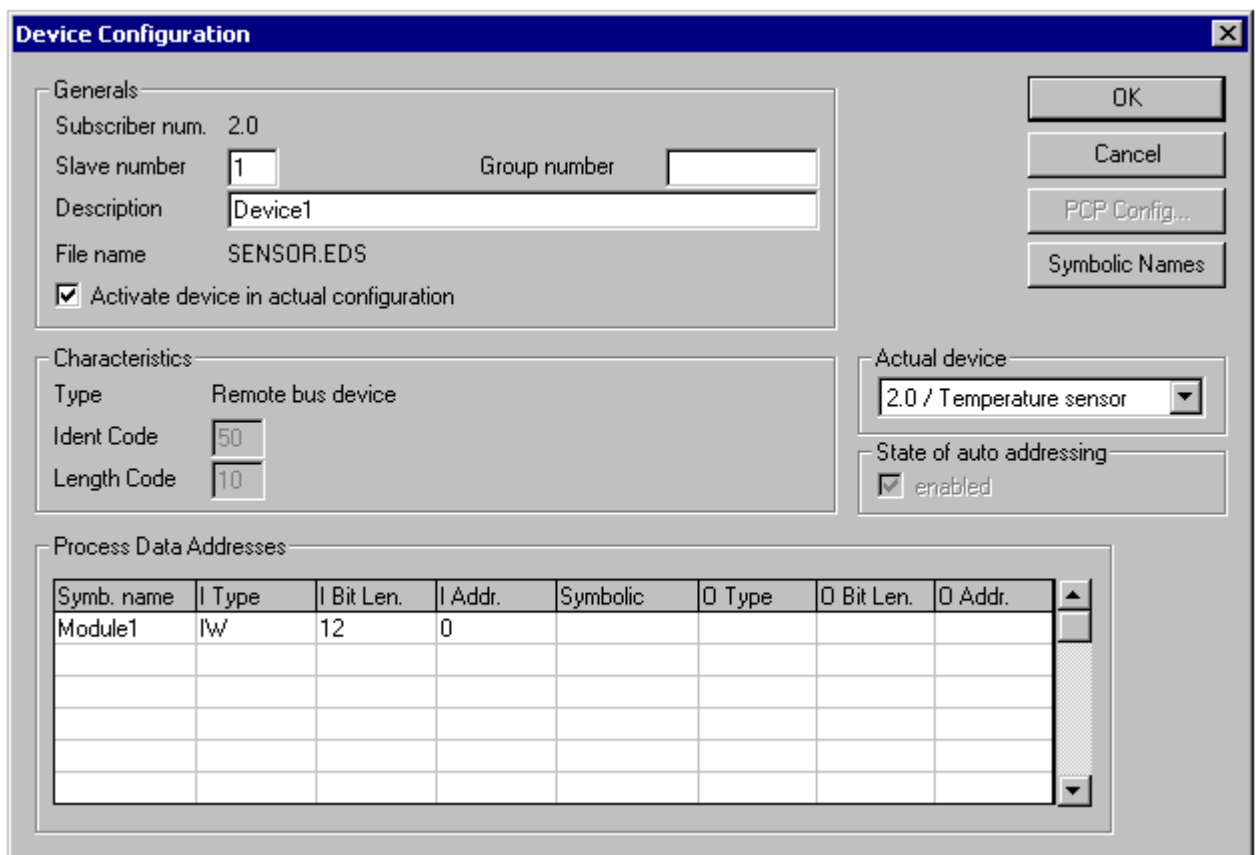


Figure 1-16: Device configuration window.bmp

Finally the system global parameters (bus parameters) have to be set. For this, mark the master in the main window and select *Settings – Bus Parameter*. The following window appears.

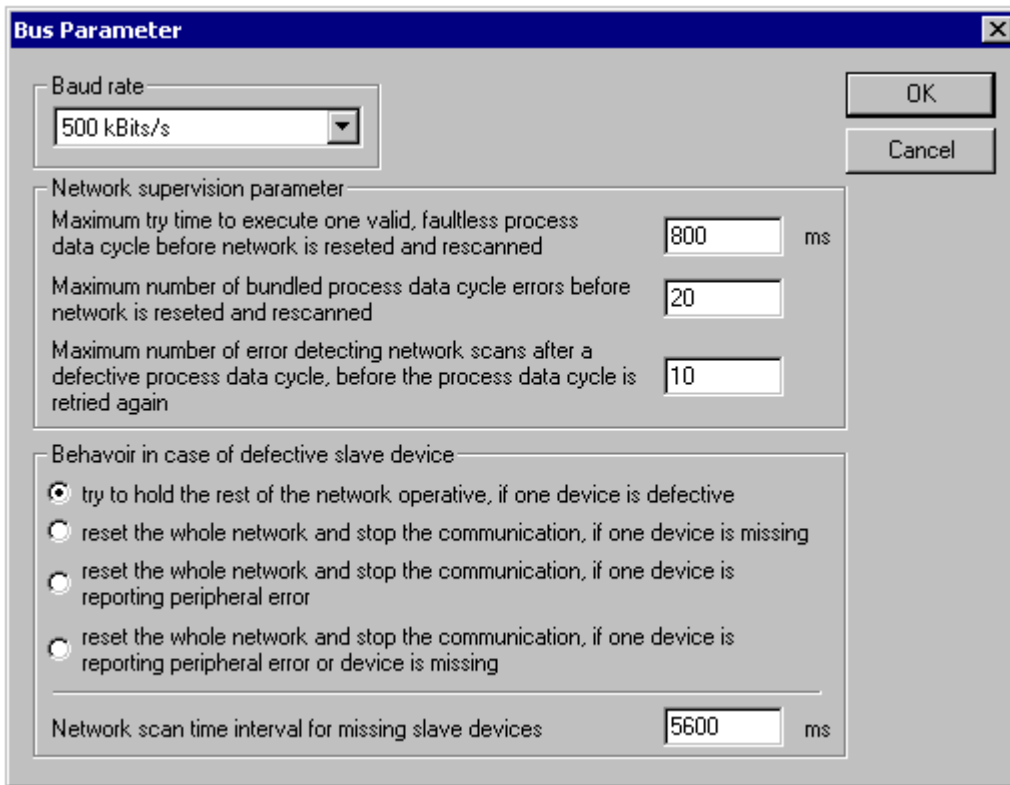


Figure 1-17: Bus parameter window.bmp

In this view different parameters can be set which influence the behavior of the master on the bus. In Interbus the default data transmission rate (baud rate) is set to 500 kbit/s (cannot be changed).

Network monitoring parameters

Maximum time interval for executing a valid process data cycle before resetting the network

Here, a time has to be set which is a multiple of 8 milliseconds, specifying the maximum time the master can use for one process data cycle. If an error occurs within a data cycle, the master first performs an ID cycle before it restarts the data cycle. This is repeated continuously and not stopped until the data cycle is either completed successfully or the monitoring time defined here is exceeded due to a repeated successive occurrence of errors. In the final error case (time is exceeded) the network will be reset. Depending on the parameterized behavior when detecting a faulty device (see below) and the network scan time interval in case of missing devices, the master stops the entire communication and attempts to reinitialize the network.

Maximum number of bundled process data cycle errors before resetting the network

This parameter determines the maximum allowed number of successive process data cycle errors. If this number is exceeded, for example due to high EMC disturbances on the Interbus line, the master's reaction depends on the parameterized behavior for the case of faulty devices and the set network scan time interval for the case of missing devices. Either the entire communication is stopped or it is attempted to re-initialize the network.

Maximum number of network scans after a failed process data cycle, before the next process data cycle is executed

If a data cycle error occurs, the master automatically starts an ID cycle in order to detect the error source in the network. If this subsequent cycle also fails, the process is repeated maximally as often as specified here. When reaching this upper limit, either the communication is stopped or it is attempted to reinitialize the network, depending on the behavior for the case of faulty devices and the set network scan time interval for the case of missing devices.

Behavior in case of a faulty device

Attempt to keep the network running in case of a faulty device

If this setting is marked, the status of connected subscribers is not considered and the network reset is disabled in case of an error. The master attempts to bring the faulty device back into operation in the time interval given by the configured network scan time interval.

Reset network and stop communication in case of a missing device

Having selected this setting, the entire communication is stopped and the whole network is reset if a device is missing during the first network cycle or during the data exchange (difference between actual configuration and reference configuration).

Reset network and stop communication if a device reports a local bus error

If at least one subscriber reports an Interbus specific module error (e.g. short-circuit at an output or undervoltage), the master stops the entire communication and resets the whole network.

Reset network and stop communication if a device reports a local bus error or a device is missing

This setting combines the two above mentioned options. The master stops the entire communication and resets the whole network if a subscriber is missing or at least one subscriber reports an Interbus specific module error during the first network cycle or while data are exchanged.

Network scan time interval in case of missing devices

This parameter sets the time interval in which missing subscribers are searched on the bus. Possible values for the network scan time interval are multiple values of 800 milliseconds or 0. The value 0 disables the function, i.e. it is not searched for missing subscribers during running operation. Otherwise missing devices are searched in the specified time interval. Note, that input and output values are not updated during the scan cycle.

For our example the predefined default values can be used. In a concrete application the parameters may be changed according to the required system behavior.

The creation of our example configuration is now completed and should be saved. In order to download the configuration data to the control, mark the master and select *Settings – Device Association*. A dialog appears used for configuring the gateway (Reference to document gateway-configuration).

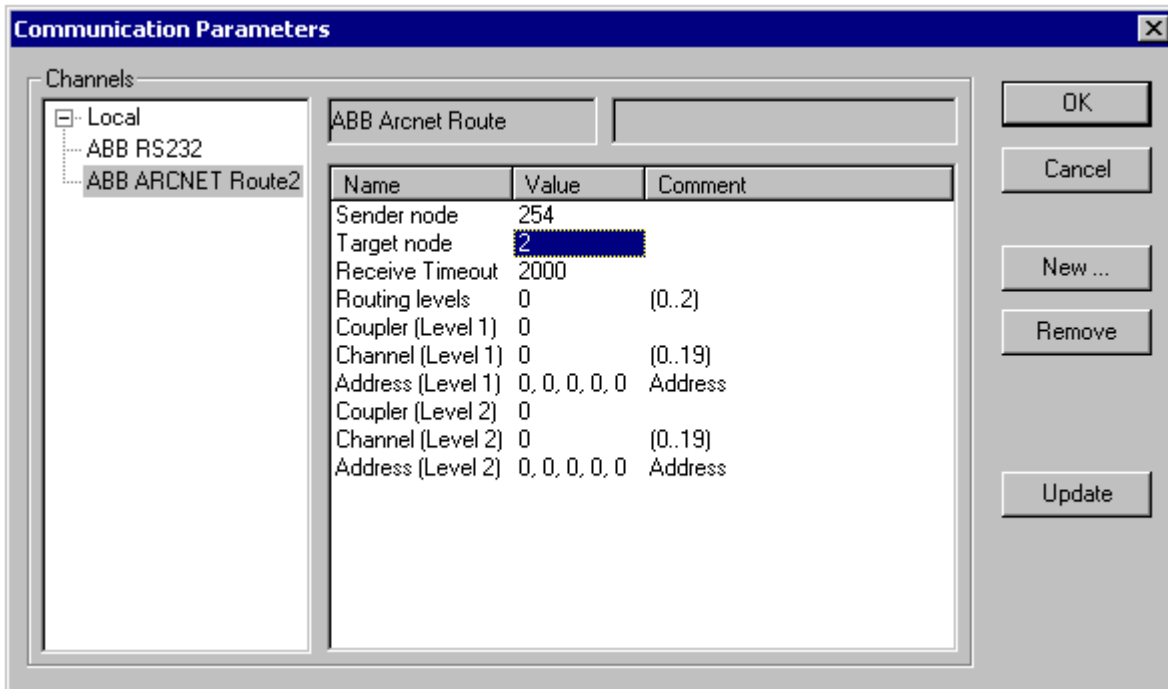


Figure 1-18: Gateway configuration for 907 FB 1131

By selecting *Configure Gateway* you can either open a channel which has already been defined or you can specify a new channel. For transmitting the configuration data the following drivers are available:

- ABB Arcnet Route
- ABB RS232 Route

Set the parameter according to the control to which the data have to be downloaded. Doing this, it is important to specify the correct slot number of the desired coupler. For the control 07 KT 97 R 169 used in this example, enter slot 2. Select *Connect* to open the set communication channel. If the connection has been established successfully, various information concerning the detected coupler are displayed in the *Device information* area and the field *Errors* contains the value 0. If no communication was possible, the field *Errors* contains a numerical code specifying the occurred error (refer to the 907 FB 1131 documentation). After the connection has been established successfully, you have to confirm the selection of the coupler by *marking it with a cross* and then clicking on *OK*. Now the previously selected channel is assigned uniquely to the opened configuration file. Now select *Online – Download* and confirm the following security dialog with *OK*. A window appears displaying the download progress. The window is closed automatically after the download is completed.

For each fieldbus coupler (except ARCNET) an own operand range is reserved in the control (Reference to documentation Systemtechnik Operanden). In this area the corresponding coupler stores data received via the bus (%I area) or the data are written in the user defined program which the coupler should send via the bus (%Q area). Beneath the transmission direction (%I, %Q) the areas are identified by the slot number of the coupler. For the control 07 KT 97 R 169 used in this example the Interbus master coupler is located in slot 2. Thus, the coupler uses the input data range %IX2.? / %IB2.? / %IW2.? and the output data range %QX2.? / %QB2.? / %QW2.?. Within these ranges the actually transmitted data have to be specified. This takes us back to the address table displayed in 907 FB 1131. For the temperature sensor (Sub. No. 2.0) an input date with the size of 12 bit and the address 0 is entered in this table. Since 12 bit are not a valid data size in 907 AC 1131, a variable with the next higher data size (WORD = 16 bit) has to be defined in the 907 AC 1131 project. Declare the variable of the data type WORD within the input operand range of the internal coupler in slot 2 (%IW2.?) with the offset address 0 BYTE in this range (means also 0 WORD (%IW2.0)). Analogous, a variable of the data type BOOL (1 bit) has to be declared for the manipulated variable of the valve (Sub. No. 1.0) which is located within the output operand range of the internal coupler in slot 2 (%QX2.?). This variable has the offset address 0 BYTE within this range (means also 0 bit (%QX2.0)).

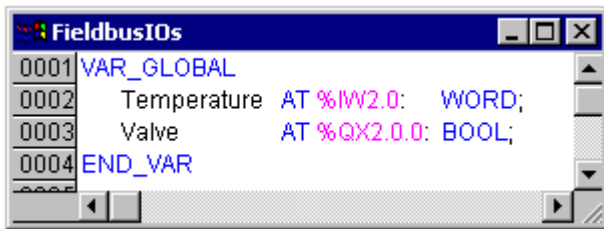


Figure 1-21: Operands.bmp

Having done these entries, the Interbus I/Os can be used within the 907 AC 1131 project.

Developing the PLC program using 907 AC 1131

Now, the actual user program can be edited. The program is intended to close the valve if the temperature is below 55 °C (manipulated variable = FALSE) and to open it if a temperature higher than 55 °C is measured (manipulated variable = TRUE). This functionality can be realized using the IEC operands GT and SEL. If the measured temperature is higher than 55 °C, TRUE is output, otherwise FALSE results.

Depending on the used temperature sensor, the read value must first be normalized accordingly. In our example we assume a temperature sensor having a measuring range of 0 °C up to 100 °C and providing the measured temperature as 12 bit analog value (value 0 = 0 °C, value 4095 = 100 °C).

So the resulting program is as follows:

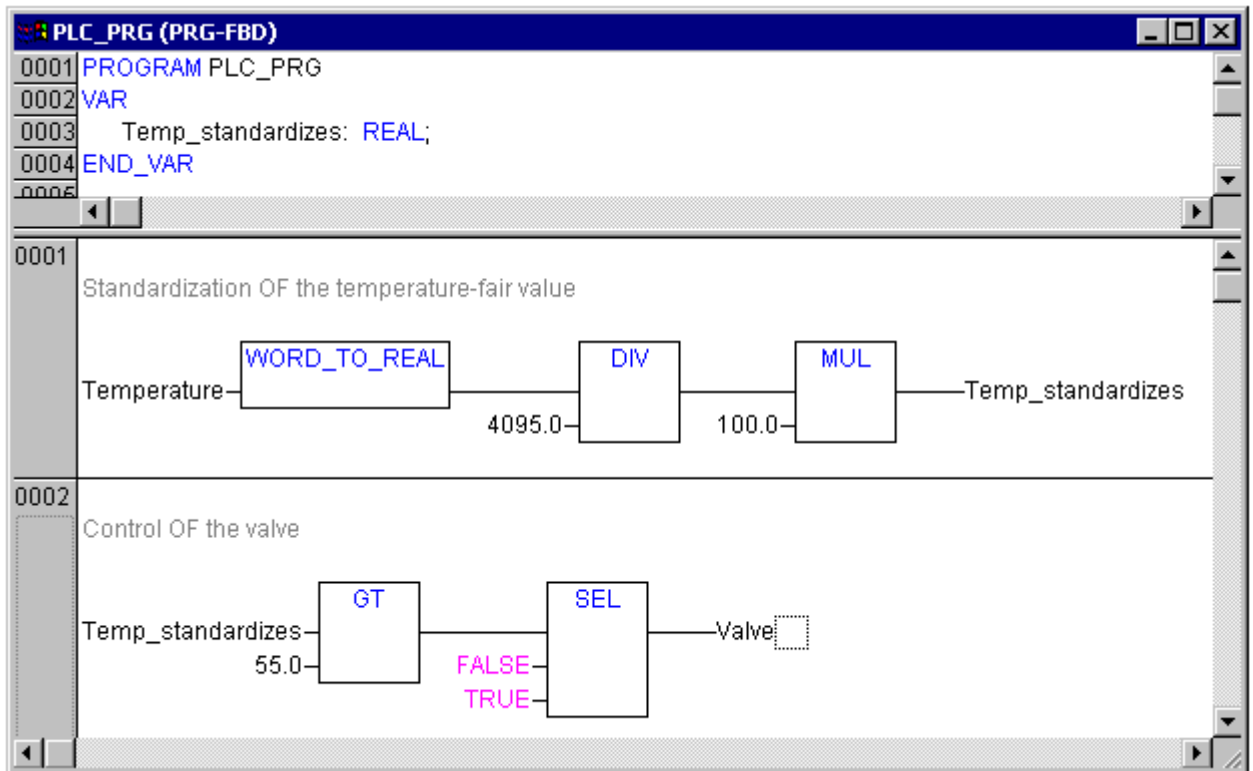


Figure 1-22: Program in 907 AC 1131

In addition, the condition of the coupler can be read out with the help of the function block IBM_STAT. If necessary, the detailed diagnostic information of the slaves can be obtained using function blocks of the type IBM_SLVDIAG. An overview of the conditions of all slaves in the system can be polled using the function block IBM_SYSDIAG.

1.7 Diagnosis

1.7.1 Status LEDs

LED	Color	Status	Meaning
RDY	Yellow	On flashes cyclic flashes irregularly Off	coupler ready bootstrap loader active hardware or system error defective hardware
RUN	Green	On flashes cyclic flashes irregularly Off	communication is running communication stopped missing or erroneous configuration no communication
STA	Yellow	On Off	<i>not used</i> <i>not used</i>
ERR	Red	On Off	Interbus failure or control in STOP status no error

Table 1-3: Status LEDs of the Interbus coupler

1.7.2 Interbus error messages

See error messages of the internal couplers.

1.7.3 Function blocks

Interbus master

- IBM_STAT: Reading the coupler status
- IBM_SLVDIAG: Reading the detailed diagnosis of a slave
- IBM_SYSDIAG: Reading the system diagnostics

1.7.4 Online diagnostics

Ref. to documentation 907 FB 1131

1.8 Further information

1.8.1 Standardization

DIN 19258

EN 50254

1.8.2 Important addresses

INTERBUS CLUB
Branch
Postfach 1108
D-32817 Blomberg

Tel.: +49 5235 342100

Fax: +49 5235 341234

Internet: <http://www.INTERBUSclub.com>

E-mail: germany@interbusclub.com

1.8.3 Terms, definitions and abbreviations used

CRC	Cyclic Redundancy Check
EDS	Electronic Data Sheet
EMC	Electromagnetic compatibility
G4	Generation 4
LWL	Optical fibre
PCP	Peripherals Communication Protocol
RC	Remote bus Check
SMC	Smart Media Card

2 Figures

Figure 1-1: Realization of the ring structure using physical point to point connections	1-1
Figure 1-2: Topology	1-3
Figure 1-3: Segmentation	1-4
Figure 1-4: Connection SUB-D 9-pin to SUB-D 9-pin	1-11
Figure 1-5: SUB-D 9-pin to MINI COMBICON 5-pin or 10-pin	1-11
Figure 1-6: SUB-D 9-pin to IP65 circular connector 9-pin	1-12
Figure 1-7: Example of a simple remote bus system	1-15
Figure 1-8: Example of a multi segment system	1-16
Figure 1-9: Planning example Interbus	1-19
Figure 1-10: Examples for the graphical valve representation in configuration, running and diagnostics mode	1-21
Figure 1-11: Examples for the graphical temperature sensor representation in configuration, running and diagnostics mode	1-21
Figure 1-12: EDS_Editor_Ventil.bmp	1-22
Figure 1-13: Inserting the master Window.bmp	1-23
Figure 1-14: Inserting a remote bus device Window.bmp	1-24
Figure 1-15: Bus_overview.bmp	1-25
Figure 1-16: Device configuration window.bmp	1-25
Figure 1-17: Bus parameter window.bmp	1-26
Figure 1-18: Gateway configuration for 907 FB 1131	1-28
Figure 1-19: Download.bmp	1-29
Figure 1-20: Address table.bmp	1-29
Figure 1-21: Operands.bmp	1-30
Figure 1-22: Program in 907 AC 1131	1-31

3 Tables

Table 1-1: Assignment	1-10
Table 1-2: Maximum line lengths	1-14
Table 1-3: Status LEDs of the Interbus coupler	1-32

Contents

1	The CANopen coupler	E 1-1
1.1	Brief overview	1-1
1.1.1	Fundamental properties and fields of application	1-1
1.1.2	Communication mechanisms	1-1
1.1.3	Network management	1-2
1.1.4	Emergency messages	1-3
1.1.5	Node guarding and heartbeat	1-4
1.1.6	Object directory	1-4
1.1.7	Identifiers	1-4
1.1.8	PDO mapping	1-5
1.1.9	EDS files	1-6
1.1.10	Features	1-6
1.2	Technical data	1-8
1.2.1	Technical data of the coupler	1-8
1.2.2	Interface specification	1-8
1.3	Connection and transfer media	1-9
1.3.1	Attachment plug for the bus cable	1-9
1.3.2	Bus terminating resistors	1-9
1.3.3	Bus cables	1-10
1.4	Possibilities for networking	1-11
1.5	CANopen implementation	1-12
1.5.1	Configuration	1-12
1.5.2	Running operation	1-12
1.5.3	Error diagnosis	1-12
1.5.4	Function blocks	1-13
1.6	Project planning example	1-14
1.7	Diagnosis	1-24
1.7.1	Status LEDs	1-24
1.7.2	CANopen error messages	1-24
1.7.3	Function blocks	1-24
1.8	Further information	1-25
1.8.1	Standardization	1-25
1.8.2	Important addresses	1-25
1.8.3	Terms, definitions and abbreviations used	1-25
2	Figures	2-I
3	Tables	3-I

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

CANopen is a standardized layer 7 protocol used for decentralized industrial automation systems based on the Controller Area Network (CAN) and the CAN Application Layer (CAL). CANopen is based on a communication profile containing the determination of basic communication mechanisms and their descriptions, such as the mechanisms used for exchanging process data in real-time or for sending alarm telegrams. This common communication profile is the basis for the various CANopen device profiles. The device profiles describe the specific functionality and/or the parameters of a device class. Such device profiles are available for the most important device classes used in industrial automation, such as digital and analog I/O modules, sensors, drives, control units, regulators, programmable controllers or encoders.

The central element of the CANopen standard is the device functionality description in an object directory (OD). The object directory is divided into one general area containing information about the device (e.g. device identification, manufacturer's name, etc.) as well as communication parameters, and the device-specific area describing the particular functionality of the device. These properties of a CANopen module are documented in the form of a standardized "electronic data sheet" (EDS file).

A CANopen network can consist of a maximum of 128 modules, one NMT master and up to 127 NMT slaves. In contrast to the typical master-slave systems (e.g. PROFIBUS systems), the meanings of the terms master and slave are different for CANopen. In operational mode all modules are independently able to send messages via the bus. Moreover, the master is able to change the operating mode of the slaves. The CANopen master is normally implemented by a PLC or a PC. The bus addresses of the CANopen slaves can be set in the range between 1 and 127. The device address results in a number of identifiers occupied by this module.

CANopen supports transmission rates of 10 kbit/s, 20 kbit/s, 50 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s. Every CANopen device has at least to support a transmission rate of 20 kbit/s. Other transmission rates are optional.

1.1.2 Communication mechanisms

CANopen distinguishes two basic mechanisms for data transmission: The fast exchange of short process data via process data objects (PDOs) and the access to object directory entries using service data objects (SDOs). The latter primarily serve for the transmission of parameters during device configuration. The transmission of process data objects is normally performed event oriented, cyclic or on request as broadcast objects.

Service data objects

Service data objects (SDOs) are used to modify object directory entries as well as for status requests. Transmission of SDOs is performed as a confirmed data transfer with two CAN objects in the form of a peer-to-peer connection between two network nodes. Addressing of the corresponding object directory entry is performed by specifying the index and the sub-index of the object directory entry. It is possible to transmit messages of unlimited length. If necessary, the data are segmented into several CAN messages.

Process data objects

For the transmission of process data the process data object (PDO) mechanism is available. Transmission of a PDO is performed unconfirmed because in the end the error-free transmission is guaranteed by the CAN link layer. According to the CAN specification a maximum of 8 data bytes can be transmitted within one PDO. In conjunction with a synchronization message, transmission as well as taking-over of PDOs can be synchronized over the entire network (synchronous PDOs). The assignment of application objects to a PDO can be adjusted according to the requirements of the individual applications by a structural description (PDO mapping) which is stored in the object directory. The transmission of process data can be performed by various methods:

Event

The PDO transmission is controlled by an internal event, e.g. by a changing level of a digital input or by an expiring device-internal timer.

Request

In this case, another bus subscriber is requesting the process data by sending a remote transmission request (RTR) message.

Synchronous

In case of a synchronous transmission, synchronization telegrams are sent by a bus subscriber. These telegrams are received by a PDO producer which in turn transmits the process data.

1.1.3 Network management

Within a CANopen network only one NMT master exists (NMT = Network management). All other modules are NMT slaves. The NMT master completely controls all modules and is able to change their states. The following states are distinguished:

Initialization

After switching-on, a node is first in the initialization state. During this phase the device application and the device communication are initialized. Furthermore, a so-called boot-up message is transmitted by the node to signalize its basic readiness for operation. After completing this phase, the node automatically changes to the pre-operational state.

Pre-operational

In this state communication with the node can be performed via service data objects (SDOs). The node is not able to perform PDO communication and does not send any emergency messages.

Prepared

In the *prepared* state a node is completely disconnected from the network. Neither SDO communication nor PDO communication is possible. A state change of the node can only be initiated by a corresponding network command (e.g. Start Node).

1.1.4 Emergency messages

Emergency messages are used to signalize device errors. Within an emergency message a code is transmitted which clearly identifies the error (specified in the communication profile DS-301 and in the individual device profiles DS-40x). The following table shows some of the available error codes. Emergency messages are automatically sent by all CANopen modules.

Emergency error code (hex)	Meaning / error cause
00xx	Error on reset or no error
10xx	General error
20xx	Current error
21xx	Error on the input side of the device
22xx	Error inside the device
23xx	Error on the output side of the device
30xx	Voltage error
31xx	Supply voltage error
32xx	Error inside the device
33xx	Error on the output side of the device
40xx	Temperature error
41xx	Ambient temperature
42xx	Temperature inside the device
50xx	Hardware error in the device
60xx	Software error in the device
61xx	Device-internal software
62xx	Application software
63xx	Data
70xx	Error in additional modules
80xx	Monitoring
81xx	Communication
90xx	External error
F0xx	Error of additional functions
FFxx	Device-specific errors

Table 1-1: Error codes in emergency messages

1.1.5 Node guarding and heartbeat

Testing the functionality of a CAN node is particularly required if the node is not continuously sending messages (cyclic PDOs). Two mechanisms can alternatively be used for monitoring CANopen nodes. When the node guarding protocol is used, the NMT master is sending messages to the available CANopen slaves which have to respond to these messages within a certain time period. Therefore the NMT master is able to detect if a node fails. Furthermore, the heartbeat protocol can be used with CANopen. With this protocol, each node is automatically sending a periodic message. This message can be monitored by each other subscriber in the network.

1.1.6 Object directory

The object directory describes the entire functionality of a CANopen device. It is organized as a table. The object directory does not only contain the standardized data types and objects of the CANopen communication profile and the device profiles. If necessary, it also contains manufacturer-specific objects and data types. The entries are addressed by means of a 16 bit index (table row, 65536 entries max.) and an 8 bit sub-index (table column, 256 entries max.). This way, objects belonging together can be easily grouped. The following table shows the structure of this CANopen object directory:

Index		Object
dec	hex	
0	0000	not used
1.....31	0001..001F	static data types
32.....63	0020..003F	complex data types
64.....95	0040..005F	manufacturer-specific data types
96.....127	0060..007F	profile-specific static data types
128.....159	0080..009F	profile-specific complex data types
160...4095	00A0..0FFF	reserved
4096...8191	1000..1FFF	communication profile (DS-301)
8192..24575	2000..5FFF	manufacturer-specific parameters
24576..40959	6000..9FFF	parameters of the standardized device profiles
40960..65535	A000..FFFF	reserved

Table 1-2: Structure of the object directory

A number of data types are defined for the objects themselves. If required, other structures (e.g. ARRAY, STRUCT) can be created from these standard types.

1.1.7 Identifiers

CANopen always uses identifiers with a length of 11 bits (standard frames). The number of available possible identifiers given by this is divided into several ranges by the pre-defined connection set. This structure is designed in a way that a maximum of 128 modules (1 NMT master and up to 127 slaves) can exist in a CANopen network. The list of identifiers is composed of some fixed identifiers (e.g. network management identifier 0) and various functional groups where each existing node supporting the corresponding function is assigned to one unique identifier (e.g. Receive PDO 1 of node 3 = 512 + node number = 515). If the pre-defined connection set is used, this prevents double assignment of identifiers.

Identifier	Function	Calculation
0	Network management (NMT)	-
1...127	not used	-
128	Synchronization (SYNC)	-
129...255	Emergency message	128 + node ID
256	Timestamp message	-
257...384	not used	-
385...511	Transmit PDO 1	384 + node ID
512	not used	-
513...639	Receive PDO 1	512 + node ID
640	not used	-
641...767	Transmit PDO 2	640 + node ID
768	not used	-
769...895	Receive PDO 2	768 + node ID
896	not used	-
897..1023	Transmit PDO 3	896 + node ID
1024	not used	-
1025..1151	Receive PDO 3	1024 + node ID
1152	not used	-
1153..1279	Transmit PDO 4	1152 + node ID
1280	not used	-
1281..1407	Receive PDO 4	1280 + node ID
1408	not used	-
1409..1535	Transmit SDO	1408 + node ID
1536	not used	-
1537..1663	Receive SDO	1536 + node ID
1664..1772	not used	-
1793..1919	NMT error (node guarding, heartbeat, boot-up)	1792 + node ID
1920..2014	not used	-
2015..2031	NMT, LMT, DBT	-

Table 1-3: Assignment of identifiers

1.1.8 PDO mapping

As already explained, all 8 data bytes of a CAN message are available for the transmission of process data. As there is no additional protocol information, the data format has to be agreed between the sending (producer) and the receiving party (consumer). This is done by the so-called PDO mapping.

If a fixed mapping is used, the process data are arranged in a pre-defined order within the PDO message. This arrangement is predetermined by the device manufacturer and cannot be modified. If variable mapping is used, the process data can be arranged as desired within the PDO message. For this purpose, the address, consisting of index and sub-index, as well as the size (number of bytes) of an object directory entry are entered into the mapping object.

1.1.9 EDS files

The characteristic properties of a CANopen module are documented in the form of an electronic data sheet (EDS file, electronic data sheet). The file completely and clearly describes the characteristics (objects) of a module type in a standardized defined and manufacturer independent format. Programs for configuring a CANopen network use the module type descriptions available in the EDS files. This strongly simplifies the planning of a CANopen system. Usually the EDS files are provided by the module's manufacturer.

1.1.10 Features

CANopen:

- Operating mode CANopen-Master
- Process image with a maximum of 57344 I/O points
- Supports min. boot-up, emergency messages and life guarding
- Supported PDO modes: Event-controlled, synchronous, cyclic and remote PDO transmission
- Integrated device profiles: CiA DS-401, CiA DS-402 and CiA DS-406

CAN (additional functionality, not necessary for pure CANopen operation):

- Support of 11 bit identifiers according to CAN 2.0A and 29 bit identifiers according to CAN 2.0B
- Transmission and reception of any CAN telegrams via function blocks in the user program

Transmission technique:

- ISO 11898, potential separated
- Transfer rates of 20 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s and 1 Mbit/s
- Bus length up to 1000 m at 20 kbit/s and up to 40 m at 1 Mbit/s
- One bus can have up to 128 subscribers (master + 127 slaves)
- 5-pin COMBICON socket for bus connection

Communication:

- Message-oriented bus access, CSMA/CA
- Predefined master slave connections
- 8 bytes of not fragmented user data, for fragmentation any size is possible
- Synchronization of inputs and/or outputs via synchronous PDOs

Protection functions:

- Message transfer with Hamming distance $HD = 6$
- CAN fault recognition mechanisms via 15 bit CRC, frame check, acknowledge, bit monitoring and bit stuffing
- Incorrect parameter settings are avoided because participants with faulty parameters are not included in the user data operation
- Adjustable behavior on subscriber failure: System continues normal operation and error is indicated at the master or entire system is stopped.
- Response monitoring of the subscribers (node guarding)

Diagnosis:

- Status indication via 4 LEDs
 - READY (yellow): The coupler is ready for operation.
 - RUN (green): Status of configuration and communication.
 - STA (yellow): Data transfer indication.
 - ERR (red): CANopen error indication.
- Extensive online diagnostics functions via 907 FB 1131
- Detailed diagnosis in the user program via function blocks

1.2 Technical data

1.2.1 Technical data of the coupler

Coupler type	CANopen master coupler in PC/104 format
Processor	16 bit processor with interrupt controller and DMA controller
Memory expansion	8 kbytes DP-RAM, 512 kbytes Flash EPROM, 128 kbytes RAM
Internal power supply with	+5 V, 650 mA
Dimensions	96 x 90 x 23 mm
CE sign	55011 Class B for Emissions, EN 50082-2 for Immunity

1.2.2 Interface specification

Interface socket	5-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	CANopen (CAN), 1 Mbaud max.
Transmission rate	Baud rates of 20 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s and 1 Mbit/s
Status indication	4 LEDs (refer to figure)
Number of subscribers	127 slaves max.

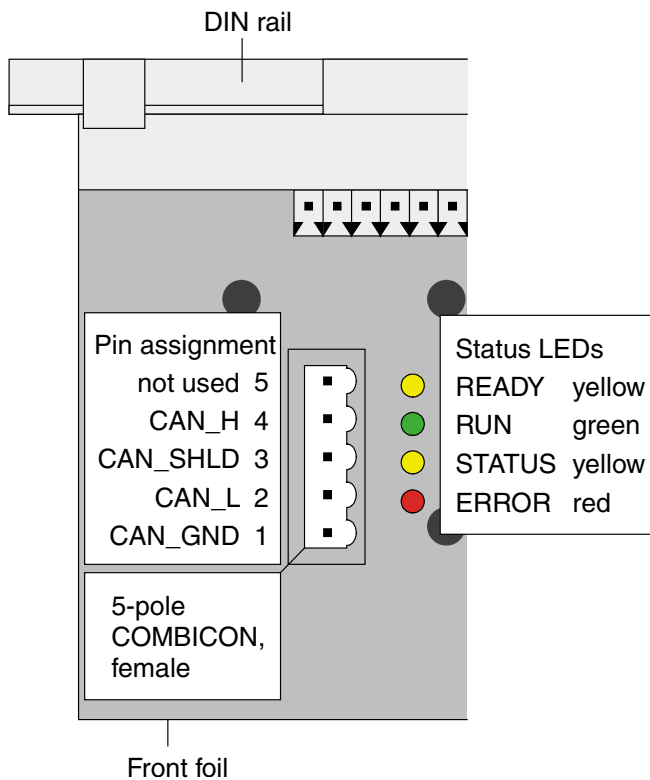


Figure 1-1: CANopen interface, connections and status LEDs

1.3 Connection and transfer media

1.3.1 Attachment plug for the bus cable

5-pin COMBICON connector

Assignment:

Pin No.	Signal	Meaning
1	CAN_GND	CAN reference potential
2	CAN_L	CAN_L bus line, receive/transmit line, low
3	CAN_SHLD	Shield of the bus line
4	CAN_H	CAN_H bus line, receive/transmit line, high
5	-	-

Table 1-4: Pin assignment of the attachment plug for the bus cable

Supplier:

e. g. COMBICON

Phoenix Contact GmbH & Co.

Flachsmarktstraße 8 - 28

D-32825 Blomberg

Phone: (+49) (0)52 35 / 3-00

Fax: (+49) (0)52 35 / 3-4 12 00

Internet: <http://www.phoenixcontact.com>

1.3.2 Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus termination resistor. The bus termination resistor is usually installed directly at the bus connector.

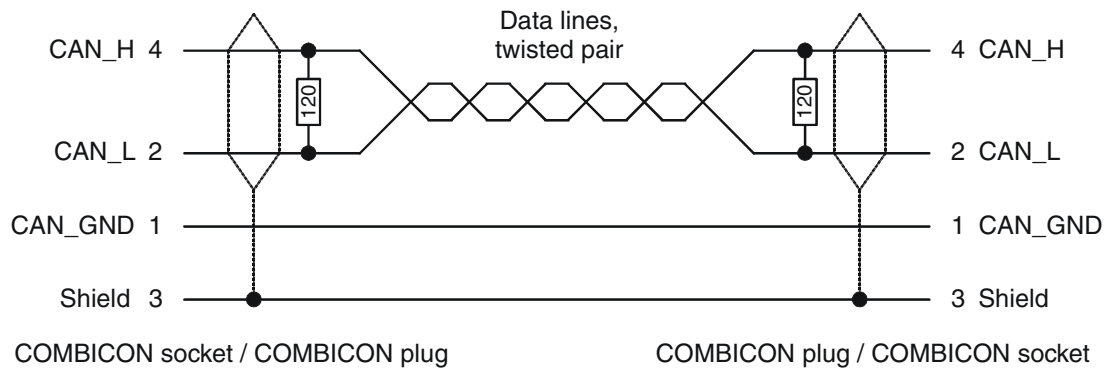


Figure 1-2: Bus terminating resistors

1.3.3 Bus cables

For CANopen only bus cables with characteristics as recommended in ISO 11898 have to be used. The requirements to the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	Bus cable			Max. baud rate [kbit/s]
	Conductor cross section [mm ²]	Line resistance [Ω /km]	Wave impedance [Ω]	
0.....40	0.25..0.34 / AWG23, AWG22	70	120	1000 at 40 m
40....300	0.34..0.60 / AWG22, AWG20	<60	120	>500 at 100 m
300....600	0.50..0.60 / AWG20	<40	120	>100 at 500 m
600..1000	0.75..0.80 / AWG18	<26	120	>50 at 1000 m

Table 1-5: Recommendations for bus cables

Supplier:

e.g. UNITRONIC® BUS CAN

U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
D-70565 Stuttgart
Phone: (+49) (0)711 7838 01
Fax: (+49) (0)711 7838 264
Internet: <http://www.lappkabel.de>

1.4 Possibilities for networking

The CANopen coupler is connected to the bus via the 5-pin COMBICON socket. For EMC suppression and protection against dangerous contact voltages, the shield of the bus line has to be connected to protective earth outside the housing. The line ends of the bus cable have to be terminated using bus terminating resistors.

Within a CANopen network, the controller with the CANopen coupler takes on the NMT master function. No other NMT master is allowed in this network. The NMT master completely controls all modules and their operational states. Up to 127 NMT slaves can be connected to an NMT master.

The CANopen master is able to:

- Change operational states of the slaves
- Parameterize the slaves (e.g. communication connections, time supervision, bus traffic)
- Configure slaves (e.g. type, number and channel operating mode)
- Read input data of the slaves
- Write output data of the slaves
- Read diagnostic data of the slaves
- Monitor the availability of the slaves
- Transmit control commands to synchronize the inputs or outputs of the slaves
- Read and write slave objects even during running operation

The CANopen coupler is as well able to:

- Transmit and receive CAN telegrams according to CAN 2.0 A (11 bit identifier) and CAN 2.0 B (29 bit identifier). (This additional functionality is not required for pure CANopen operation.)

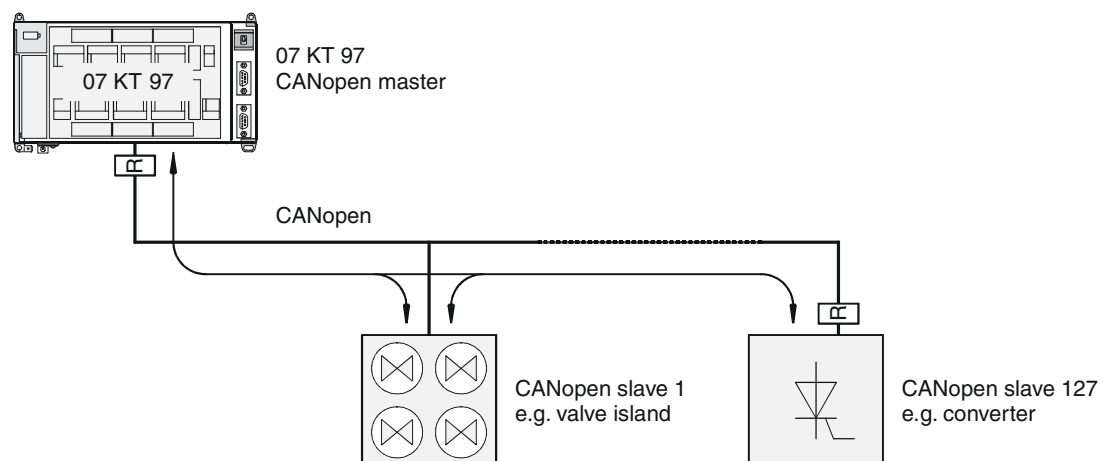


Figure 1-3: Example of a CANopen system using a 07 KT 97 as CANopen master

1.5 CANopen implementation

1.5.1 Configuration

The CANopen master coupler is configured via the PC using the 907 FB 1131 configuration software for field bus couplers (see documentation 907 FB 1131 and project planning example). Configuration data created using 907 FB 1131 are not assigned directly to a project. Thus, the resulting file has to be downloaded separately to the controller (additionally to a 907 AC 1131 project) where it is stored in the Flash memory.

If the user defined project is written to SMC, the configuration data are automatically saved, too.

1.5.2 Running operation

The CANopen protocol is automatically processed by the coupler and the operating system of the controller. The coupler is only active on the bus if it was correctly initialized before and if the user program runs. No function blocks are necessary for exchanging process data via CANopen. Special CAN and CANopen functions can be realized using the function blocks of the CANopen library.

The coupler starts the communication via CANopen after the user defined program is started and attempts to initialize the planned slaves. After a successful initialization, the slave performs the process data exchange. The I/O data exchange with the slaves is performed automatically.

If the user program is stopped, the coupler shuts down the CANopen communication in a controlled manner.

1.5.3 Error diagnosis

CANopen communication errors are always indicated by the LEDs next to the bus connector. A malfunction of the CANopen driver or the coupler itself is indicated via the FKx error flags and the corresponding LEDs (refer to the error tables of the internal couplers). Furthermore the CANopen library provides different function blocks which allow detailed error diagnosis (refer to libraries CANopen_S90_Vxx.LIB and Coupler_S90_Vxx.LIB).

1.5.4 Function blocks

Libraries:

CANopen_Master_S90_Vxx.lib and Coupler_S90_Vxx.LIB

General:

- CANopen_INFO - Reading of coupler information

CANopen master:

Status / Diagnosis

- COM_NODEDIAG - Reading the detailed diagnosis of a slave
- COM_STAT - Reading the CANopen coupler status
- COM_SYSDIAG - Displaying status information of all slaves
- COM_RES_ERR - Resetting the internal error indications and counters

SDO

- COM_SDO_READ - Reading the value of a slave object
- COM_SDO_WRITE - Writing the value of a slave object

NMT

- COM_NMT - Changing the operational state of one or all slaves

CAN 2.0 A:

- CAN_REC_2A - Receiving CAN 2.0 A telegrams (11 bit identifier)
- CAN_REC_FILTER_2A - Setting up receive filters for CAN 2.0 A telegrams
- CAN_SEND_2A - Transmitting CAN 2.0 A telegrams (11 bit identifier)
- IDENT_CAN2A_TO_WORD - Generating a general CAN 2.0 A telegram header
- WORD_TO_IDENT_CAN2A - Splitting a general CAN 2.0 A telegram header
- IDENT_CANopen_TO_WORD - Generating a CANopen-specific CAN 2.0 A header
- WORD_TO_IDENT_CANopen - Splitting a CANopen-specific CAN 2.0 A header

CAN 2.0 B:

- CAN_REC_2B - Receiving CAN 2.0 B telegrams (29 bit identifier)
- CAN_SEND_2B - Transmitting CAN 2.0 B telegrams (29 bit identifier)

1.6 Project planning example

Below, a planning example for the 07 KT 98 R168 used as CANopen master is shown. Beneath the CANopen master 07 KT 98, a valve and a temperature sensor are available operating as CANopen slaves in the network. So, the resulting configuration is as follows:

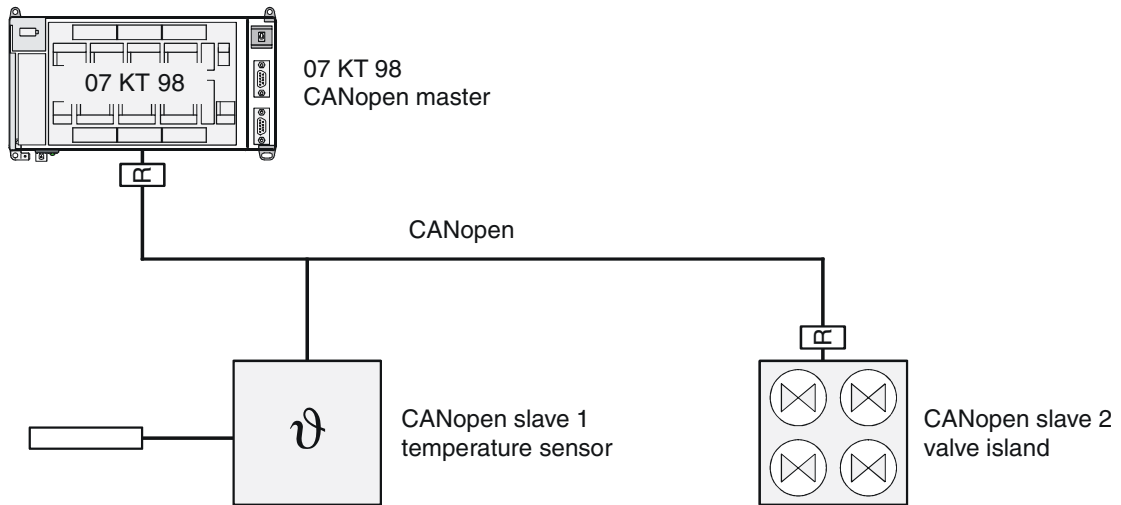


Figure 1-4: CANopen example configuration


The controller used as CANopen master reads the temperature values measured by the sensor. Depending on this value, a control signal is generated and sent to the valve. The following node addresses are agreed: Temperature sensor = 1, valve = 2. It is not necessary to assign a node address for the CANopen master 07 KT 98.

Configuration using 907 FB 1131:

Prior to taking the master PLC into operation it is recommended to set up the slaves first. Refer to the manufacturer's documentation for the procedure for setting up the sensor and the valve. Such devices often have DIP switches or rotary switches used to set the node address as well as the transmission rate (if not determined automatically).

After the CANopen slaves have been configured, the configuration of the master can be done. For that purpose start the configuration software 907 FB 1131. If the slaves detailed in this example are used for the first time, you first have to import the EDS files (refer to 907 FB 1131 documentation). In our example, this is not required, because we are alternatively using the provided default ESD file (STANDARD.EDS).

In the *File* menu select *New*. A window appears, listing all available field bus systems. In this window select *CANopen* and confirm with *OK*. The main window now displays an empty bus strand. Save the file under any name in any directory by selecting *File – Save as...* (e.g. *CANopen_Master_Sample*). The extension *.co* (**CANopen**) is automatically attached to the file name.

Now insert the 07 KT 98 as CANopen master. For that purpose click on the button *Insert master* . Select *07 KT 98-COM* in the *Available masters* list and click on the *Add >>* button. After this, the 07 KT 98 is inserted as CANopen master into the *Selected masters* list. Now, enter a unique device description (e.g. SampleMaster) and confirm with *OK*.

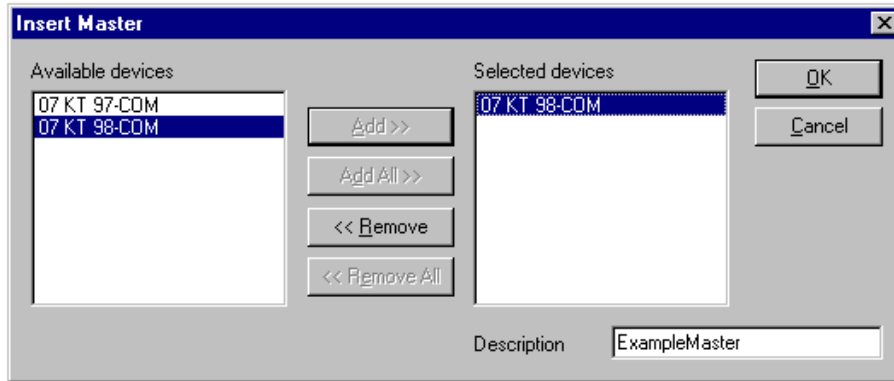



Figure 1-5: Inserting a master

Now insert the slaves in the same way. Select the button *Insert node*  and then click into a main window area below the previously inserted master. Select *Profile 401 standard EDS* in the *Available devices* list and click on the *Add >>* button. If the desired entry is not visible, set the *Node filter* for *Vendor* and *Profile* to *All*. After adding the device, it is available as a CANopen node in the *Selected devices* list. Now specify the desired node address and define a unique device description. As this device represents the temperature sensor in our example, enter the node address 1 and for example the description "Temperature sensor". Focus again the list containing the available devices in order to select the second slave. Select once again the entry *Profile 401 standard EDS* and apply your selection with *Add>>*. For this device set the node address 2 and enter the description "Valve". Confirm with *OK*.

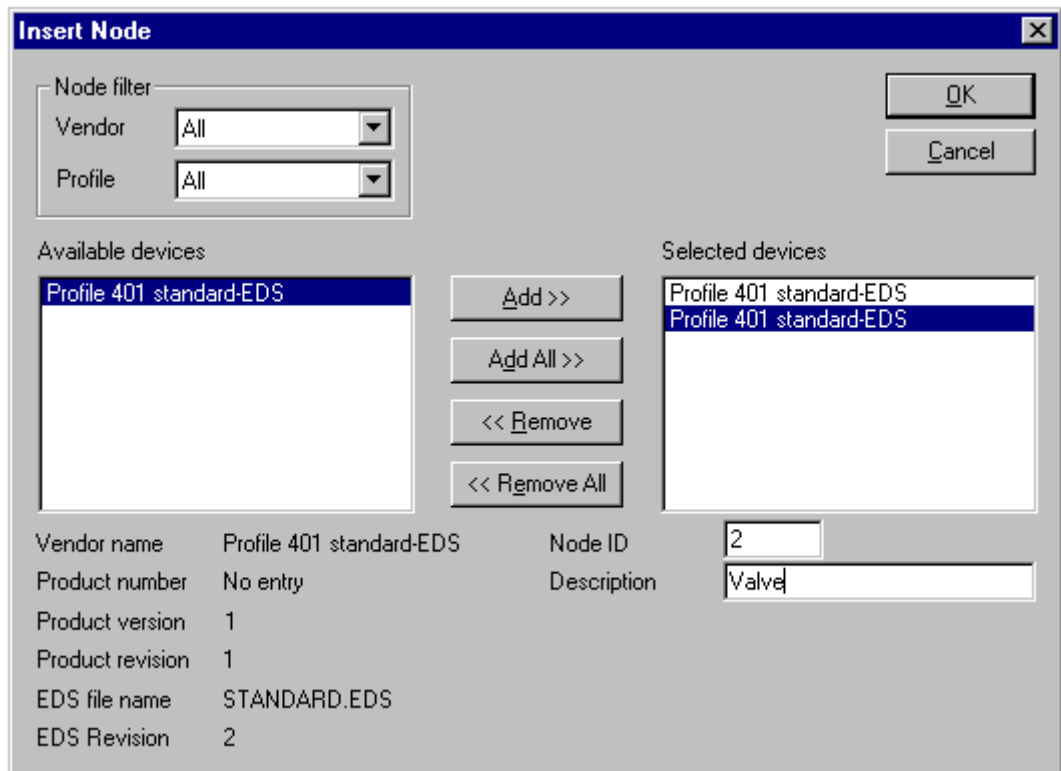


Figure 1-6: Inserting a node

Thus, the following network configuration is obtained:

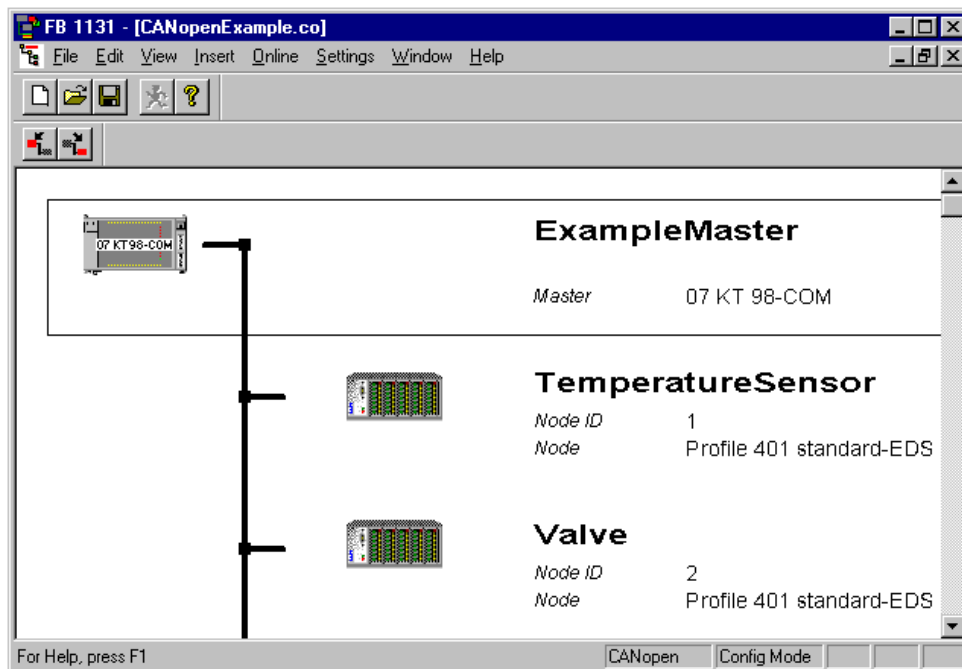


Figure 1-7: Bus overview

In the previous steps the subscribers connected to the bus were described. Now, the I/O data (PDOs) to be transmitted between the various subscribers and the corresponding transmission mechanisms (PDO properties) have to be defined. For that purpose first select the temperature sensor (refer to the figure shown above). In the marked area press the right mouse button and select *Node configuration* in the appearing context menu. The following window appears.

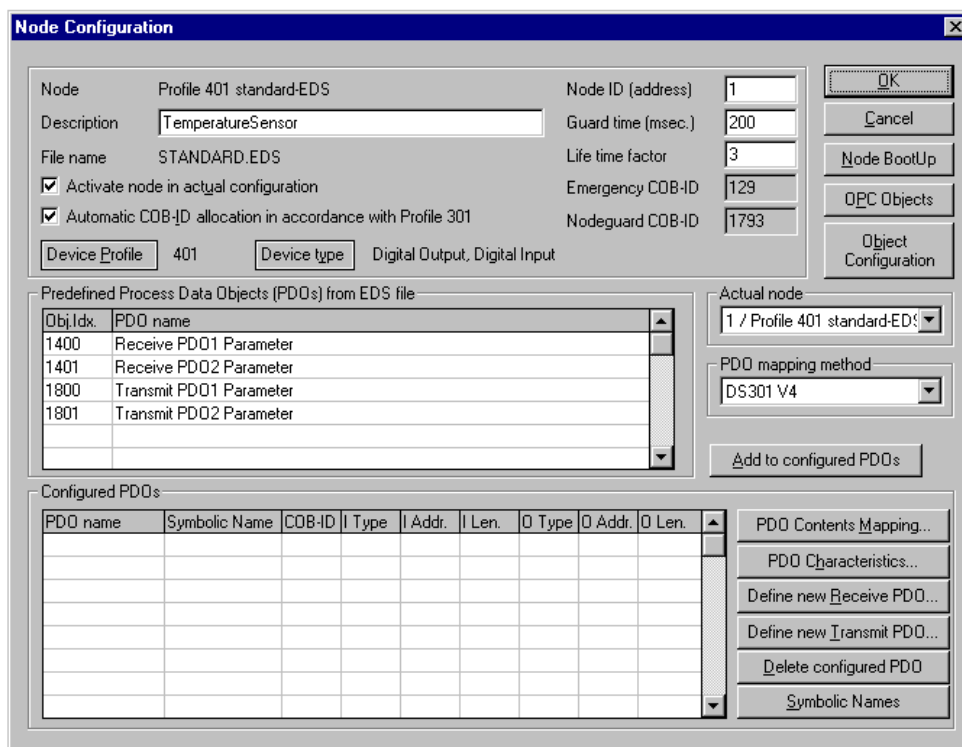


Figure 1-8: Node configuration

In this view, the number of process data to be transmitted/received as well as the corresponding transmission event control are configured.

In this example the master should read the current temperature from the slave. The *Available predefined process data objects (PDOs) list from the EDS file* displays all I/O data types which are supported by the device to be configured. Note, that the designation of the transmission direction (receive/transmit) always reflects the CANopen view (not the process view). This means that receive PDOs are data, a node receives via the bus and then outputs to the process on-site. Transmit PDOs are data which are transmitted by a node via the bus and processed by the recipient (e.g. master). Assuming that the slave detects the temperature from the process as a byte value and transmits this byte to the master, you have to select the entry *Transmit PDO 1 Parameter* in this list and add this selection to the configured PDOs by clicking on *Add to configured PDOs*. This opens the following dialog where you can define the properties of the selected PDO (transmission mechanism).

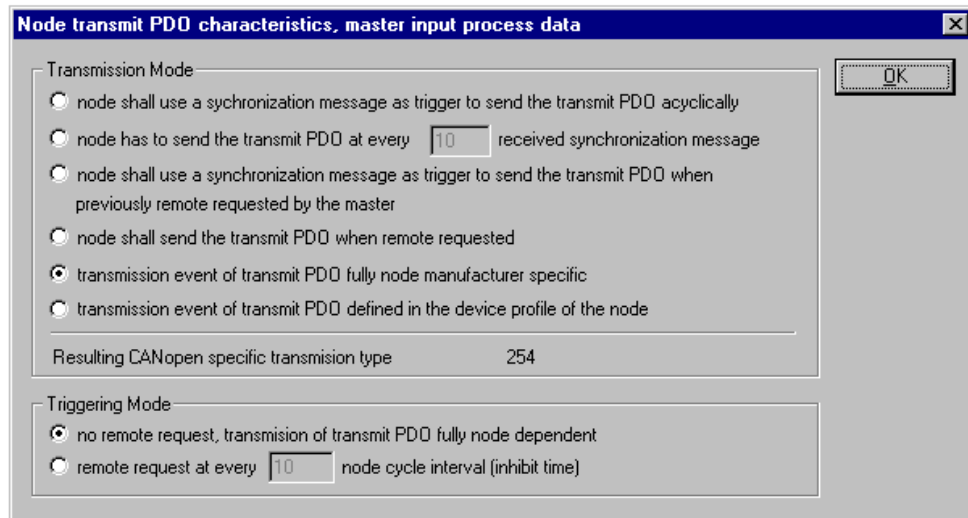


Figure 1-9: PDO properties

Which data transmission mechanism is suitable and should therefore be selected, depends on the current application. Assuming that the measured temperature is changing very slowly in our example, it is sufficient to read the sensor value periodically in longer intervals controlled by the master. For this purpose select the option *node shall send the transmit PDO when remote requested* in the *Transmission Mode* group box. This automatically also selects the option *remote request at every X node cycle interval (inhibit time)* in the *Triggering Mode* group box. For the number of node cycles enter 1000 and confirm with *OK*. The selected PDO is then displayed in the lower list of *configured PDOs*. The PDO properties just defined can be modified at any time by highlighting the PDO in the list of *configured PDOs* and then clicking on the *PDO Characteristics* button.

For the standard node used in this example the PDO first has a predefined length (*len.*) of 8 bytes. For transmitting the temperature only 1 byte is required. Therefore, double click on the PDO in the *configured PDOs* list. This opens the dialog *PDO Contents Mapping Object Index 1A00* which shows two lists. The *Mapped Object dictionary* assigned to the selected PDO are a selection of the configurable objects in the EDS file. The entries *Sub. Idx. 2 to 8* are not necessary because, as already mentioned, only one byte is required for transmitting the temperature.

Delete the unnecessary sub-index entries by highlighting the corresponding entry and then clicking on the button *Delete mapped Object*. Confirm your entries with *OK*.

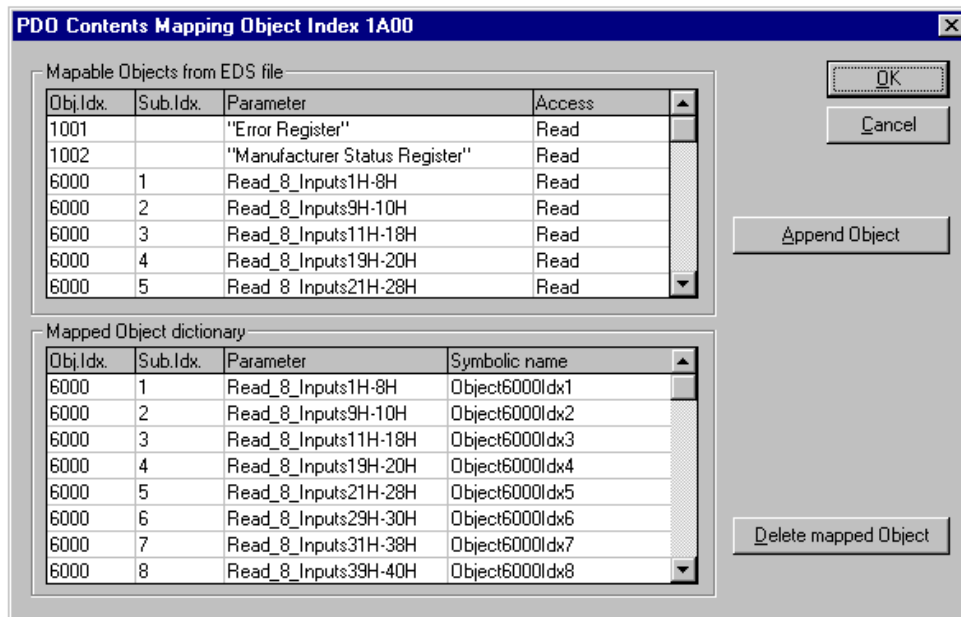


Figure 1-10: Objects

The resulting node configuration is as follows:

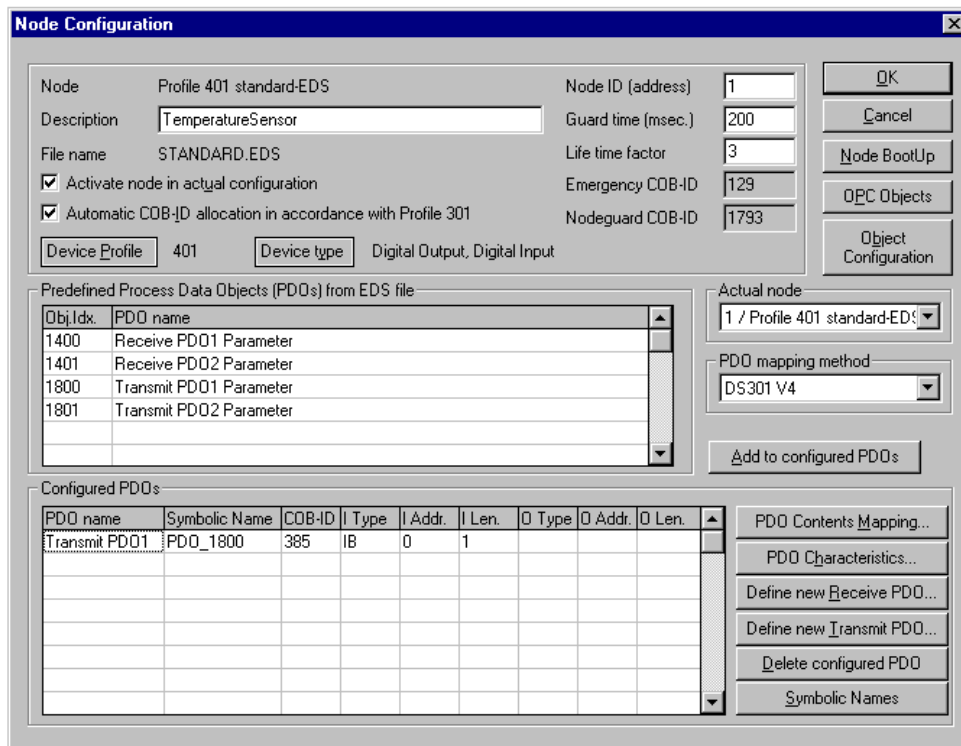


Figure 1-11: Node configuration

In our example the further possible settings and objects are not described in detail. For their meanings please refer to the 907 FB 1131 documentation and the corresponding module descriptions.

The I/O configuration of the temperature sensor is now completed and can be confirmed with *OK*.

In the next step the valve has to be configured accordingly as second slave. The valve obtains its manipulated variable as a byte value via the bus and passes it to the process. Because the valve receives the data from the master, a receive PDO has to be configured. For this purpose, first proceed as described for configuring the temperature sensor: Select the valve in the bus overview (refer to the figure above), right click in the marked area and then select *Node configuration* in the appearing context menu. This opens again the dialog for configuring the node. From the list *Predefined process data objects (PDOs) from the EDS file* select the entry *Receive PDO1 parameter* and add it to the *configured PDOs* by clicking on the *Add to configured PDOs* button. In the appearing dialog you can now define the properties of the selected PDO. In order to avoid unnecessary bus load, specify that the value shall only be transmitted, if it has changed since the last transmission. For this purpose select the Transmission Mode *receive PDO transmission Triggering Mode dependent only* and the *Triggering Mode event driven, PDO transmitted when data has changed* and then confirm with OK.

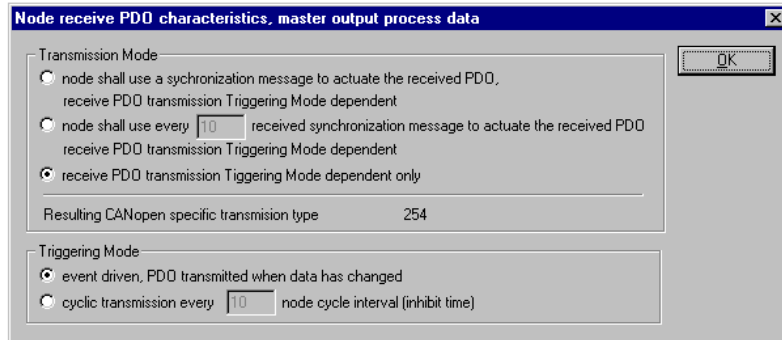


Figure 1-12: PDO properties

The transmission of the control value to the valve also requires only 1 byte (as the temperature sensor). For the standard node used in this example the corresponding PDO first has a predefined length (*Obj Len.*) of 8 bytes. Therefore, delete all configured objects of the PDO which are not required as described for the temperature sensor.

This results in the following CANopen slave configuration for the valve.

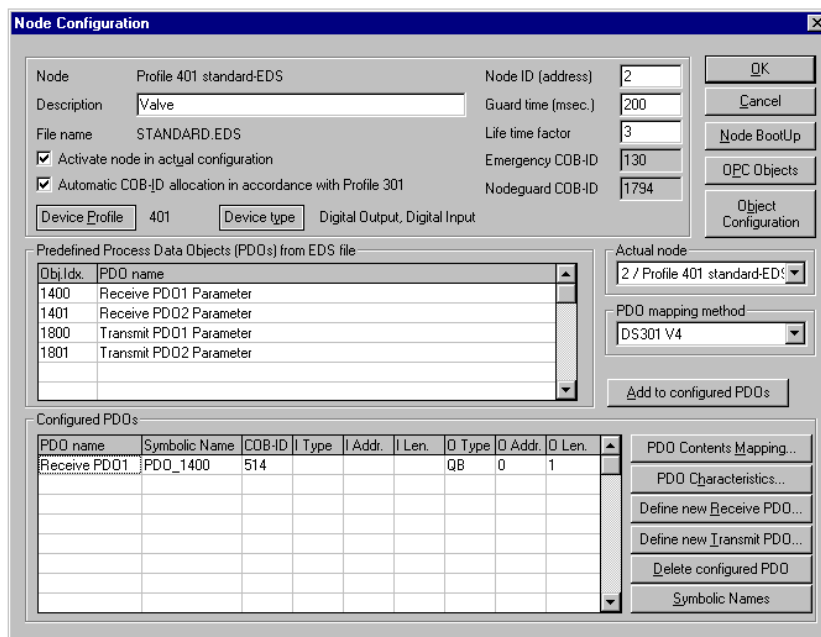


Figure 1-13: Node configuration

Finally the bus parameters have to be set. For this, highlight the master in the main window and select *Settings - Bus Parameters*. From the appearing dialog select a baud rate which is supported by all bus nodes. If the devices do not have automatic baud rate detection, the transmission rate has to be set to the same value for all nodes (refer to the descriptions at the beginning of this example). Select the desired baud rate and confirm with *OK*. In our example the further possible settings in this dialog are not described in detail. For their meanings please refer to the 907 FB 1131 documentation. The configuration is now completed and should be saved once more.

In order to download the configuration data to the controller, mark the master and select *Settings – Device Assignment*. The dialog for configuring the gateway appears (refer to System technology of the basic units / Programming and testing).

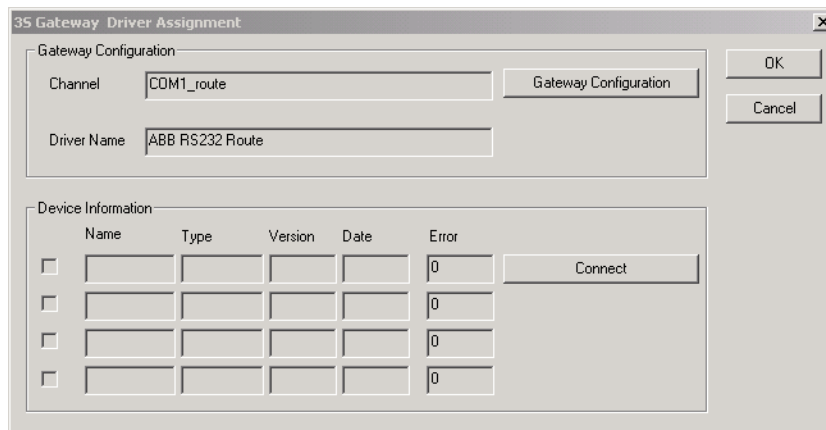


Figure 1-14: Gateway

By selecting *Gateway Configuration* you can either open a channel which has already been defined or you can specify a new channel. For transmitting the configuration data the following drivers are available:

- ABB Arcnet Route
- ABB SL97
- ABB RS232 Route

Set the parameters according to the controller to which the data have to be downloaded. In our example we are assuming that the controller to which the configuration data have to be sent is directly connected to the PC via the serial interface. In this case the driver (ABB RS232 Route) and the parameters have to be set as shown in the following figure.

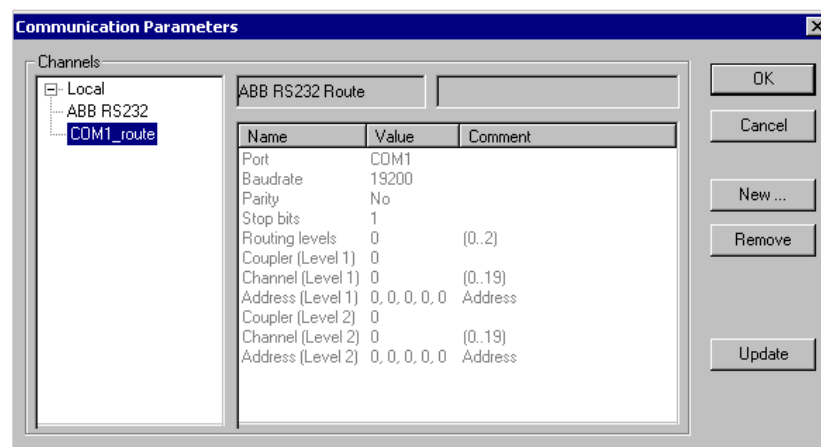


Figure 1-15: Communication parameters

Select **OK** to open the set communication channel. 907 FB 1131 then tries to establish a connection to the set controller. The result is displayed in the *Device Information* area. If general connection errors occur, the field *Errors* in the first line of the table displays a corresponding error identifier (refer to the 907 FB 1131 documentation). All other fields are empty and gray. No coupler can be selected.

After establishing the connection successfully, the *Device Information* area displays various information about the couplers installed in the concerning controller which match the present configuration data file and are ready for downloading. Unsuitable couplers or couplers which are not ready, are signalized by a corresponding error message.

The controller used for this example is a 07 KT 98 R168. The CANopen coupler is located in the internal coupler slot 2. If the controller is in STOP state, the configuration data can be transmitted to the coupler. The result of the connection establishment should read as follows.

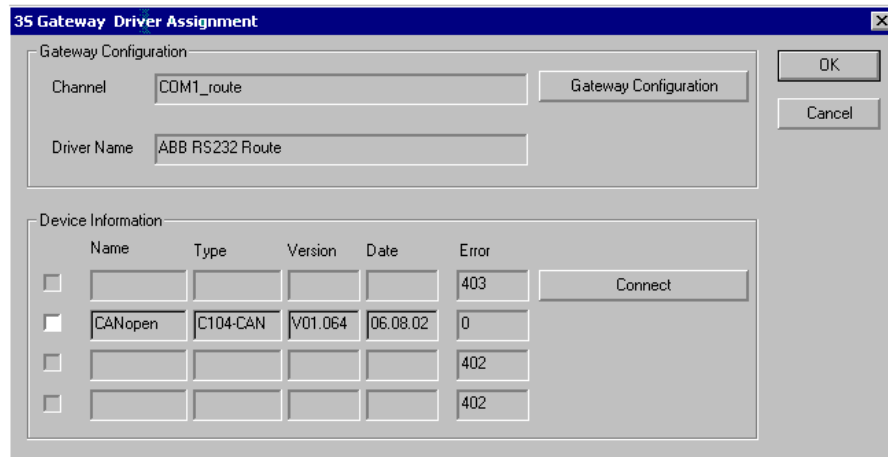


Figure 1-16: Driver assignment

The ARCNET coupler in slot 1 is a coupler type which is not suitable for a download of the configuration data (error 403). The CANopen coupler in slot 2 is ready and the only suitable coupler which can be selected in this controller. Slots 3 and 4 are not supported by the concerning controller (error 402). Select the CANopen coupler by marking the checkbox on the left side of the entry and confirm with **OK**. Now the selected channel is uniquely assigned to the opened configuration file.

Select *Online - Download* from the menu and confirm the following safety inquiry with **Yes**. A window appears displaying the download progress. The window is automatically closed after the download is completed.

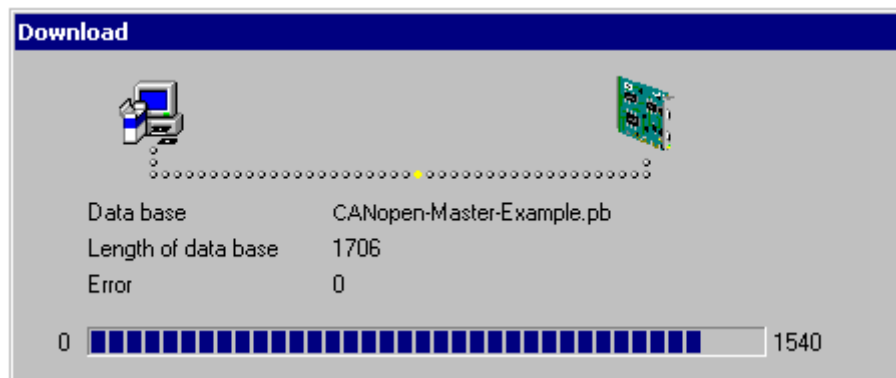


Figure 1-17: Download

Importing the configuration to 907 AC 1131

Prior to the creation of the actual 907 AC 1131 project, it has to be determined in which areas the CANopen I/O data should be stored (refer to 907 FB 1131 documentation / Importing the configuration to 907 AC 1131). Highlight the master and then select *View – Address Table* from the menu. The appearing window displays a list of all slaves (nodes) communicating with the master as well as all configured communication parameters.

Node ID	Device	Obj. Idx	Parameter	COB-ID	I Type	I Adr.	I Len.	O Type	O Adr.	O Len.
1	Profile 401	1800	Transmit PDO1	385	IB	0	1			
2	Profile 401	1400	Receive PDO1	514				QB	0	1

Figure 1-18: Address table

With the help of this table, the appropriate type, size and address of all I/O data have to be determined. These information have to be imported into the related 907 AC 1131 project. For that purpose start the 907 AC 1131 software. Create a new project for a controller 07 KT 98 (refer to 907 AC 1131 documentation) and save it as *CANopen_Master_Example*. Select the *Resources* view and create a *New Object* named *FieldbusIOs* in the *Global Variables*.

For each internal field bus coupler (except ARCNET) an own operand area is reserved in the controller (refer to documentation System technology of the basic units / Operands) where the corresponding coupler stores the data received via the bus (%I area) or where the data the coupler has to send via the bus (%Q area) have to be stored in the user defined program. Apart from the transmission direction (%I, %Q) the areas are identified by the slot number of the coupler. For the controller 07 KT 98 R 168 used in this example the CANopen master coupler is located in slot 2. Thus, the coupler uses the input data range %IX2.? / %IB2.? / %IW2.? and the output data range %QX2.? / %QB2.? / %QW2.?. Within these ranges the actually transmitted data have to be specified. This takes us back to the address table displayed in 907 FB 1131. For the temperature sensor (node 1) an InputByte (IB) with the size 1 and address 0 is entered in this table. Accordingly, a variable of the data type BYTE has to be declared in the 907 AC 1131 project which is located within the input operand range of the internal coupler in slot 2 (%IB2.?). The variable has the offset address 0 BYTE (%IB2.0) within this range. Analogous, a variable of the data type BYTE has to be declared for the manipulated variable of the valve (node 2) which is located in the output operand range of the internal coupler in slot 2 (%QB2.?). The variable has the offset address 0 BYTE (%QB2.0) within this range.

Address	Variable Name	Declaration
0001	VAR_GLOBAL	
0002	Temperature	AT %IB2.0: BYTE;
0003	Valve_position	AT %QB2.0: BYTE;
0004	END_VAR	

Figure 1-19: Operands

Having done these entries, the CANopen I/Os can be used within the 907 AC 1131 project.

Developing the PLC program using 907 AC 1131

Now, the actual user program can be created. The program is intended to close the valve if the temperature is below 55°C (manipulated variable = 0(%)) and to open it completely if a temperature higher than 55°C is measured (manipulated variable = 100(%)). This functionality can be realized using the IEC operands GT and SEL. If the measured temperature is higher than 55°C the value 100 is output, otherwise a 0 results.

Depending on the used temperature sensor, the read value must first be normalized accordingly. In our example we assume a temperature sensor having a measuring range of 0°C up to 100°C and providing the measured temperature as an 8 bit value in the form of one BYTE (value 0 = 0°C, value 255 = 100°C).

So the resulting program is as follows:

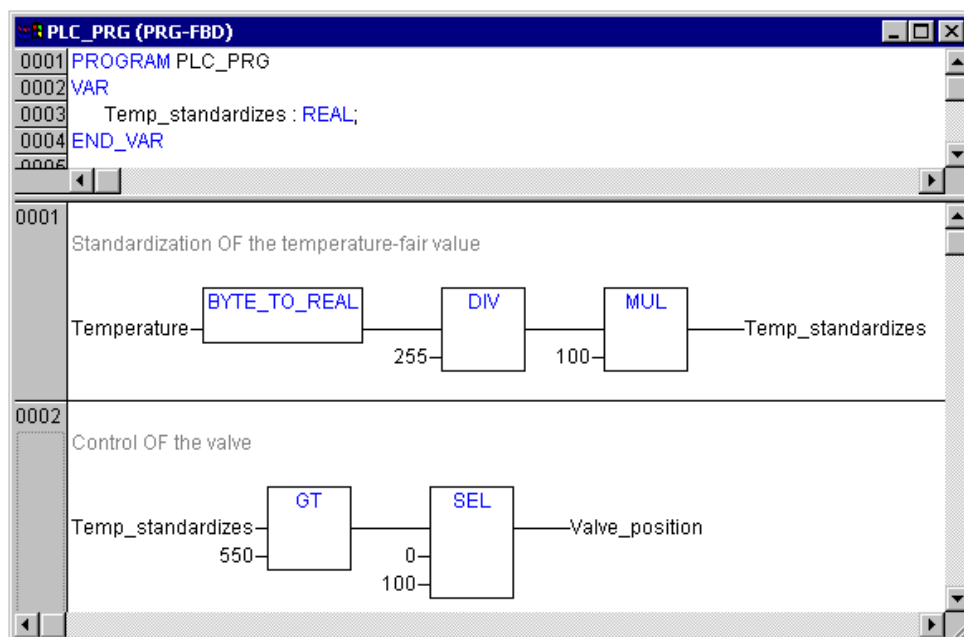


Figure 1-20: Program

In addition, the condition of the coupler can be read with the help of the function block COM_STAT. If necessary, the detailed diagnostic information of the slaves can be polled using function blocks of the type COM_NODEDIAG and an overview of the conditions of all slaves in the system can be obtained using the function block COM_SYSDIAG.

1.7 Diagnosis

1.7.1 Status LEDs

LED	Color	Status	Meaning
RDY	yellow	On	coupler ready
		flashes cyclic	bootstrap loader active
		flashes irregularly	hardware or system error
		Off	defective hardware
RUN	green	On	communication is running
		flashes cyclic	communication stopped
		flashes irregularly	missing or faulty configuration
		Off	no communication
STA	yellow	On	coupler is transmitting data
		Off	coupler is not transmitting data
ERR	red	On	CANopen error
		Off	no error

Table 1-6: Status LEDs

1.7.2 CANopen error messages

The CANopen error messages are listed in section 'Error messages of the internal couplers'.

1.7.3 Function blocks

CANopen master:

- COM_NODEDIAG - Reading the detailed diagnosis of a slave
- COM_STAT - Reading the CANopen coupler status
- COM_SYSDIAG - Displaying status information of all slaves
- COM_RES_ERR - Resetting the internal error indications and counters

Online diagnosis:

See documentation for the field bus configurator 907 FB 1131.

1.8 Further information

1.8.1 Standardization

BOSCH CAN specification - version 2.0, part A & part B

ISO 11898

CiA DS 201 V1.1 - CAN Application Layer

CiA DS 301 V3.0 - CAL based Communication Profile for Industrial Systems

CiA DS 301 V4.02 - CANopen Application Layer and Communication Profile

CiA DS 401 V2.1 - CANopen Device Profile Generic I/O modules

CiA DS 402 V2.0 - CANopen Device Profile Driver and Motion Control

CiA DS 406 V3.0 - CANopen Device Profile Encoder

1.8.2 Important addresses

CAN in Automation (CiA)
Am Weichselgarten 26
D-91058 Erlangen
Germany

Phone: +49 9131 69086-0

Fax: +49 9131 69086 -79

Internet: <http://www.can-cia.de>

1.8.3 Terms, definitions and abbreviations used

CAL	CAN Application Layer
CAN	Controller Area Network
CiA	CAN in Automation international users and manufacturers group e.V.
DLC	Data Length Code
EDS	Electronic Data Sheet
ISO	International Standardisation Organisation
NMT	Network Management
OD	Object Directory
PDO	Process Data Object
RTR	Remote Transmission Request
SDO	Service Data Object

2 Figures

Figure 1-1: CANopen interface, connections and status LEDs	1-8
Figure 1-2: Bus terminating resistors.....	1-9
Figure 1-3: Example of a CANopen system using a 07 KT 97 as CANopen master.....	1-11
Figure 1-4: CANopen example configuration	1-14
Figure 1-5: Inserting the master.....	1-15
Figure 1-6: Inserting a node.....	1-15
Figure 1-7: Bus overview.....	1-16
Figure 1-8: Node configuration	1-16
Figure 1-9: PDO properties.....	1-17
Figure 1-10: Objects	1-18
Figure 1-11: Node configuration	1-18
Figure 1-12: PDO properties.....	1-19
Figure 1-13: Node configuration	1-19
Figure 1-14: Gateway	1-20
Figure 1-15: Communication parameters	1-20
Figure 1-16: Driver assignment	1-21
Figure 1-17: Download.....	1-21
Figure 1-18: Address table.....	1-22
Figure 1-19: Operands	1-22
Figure 1-20: Program.....	1-23

3 Tables

Table 1-1: Error codes in emergency messages.....	1-3
Table 1-2: Structure of the object directory.....	1-4
Table 1-3: Assignment of identifiers.....	1-5
Table 1-4: Pin assignment of the attachment plug for the bus cable.....	1-9
Table 1-5: Recommendations for bus cables.....	1-10
Table 1-6: Status LEDs.....	1-24

Contents

1	Ethernet and protocols	F-1-1
1.1	History	1-1
1.1.1	History of Ethernet	1-1
1.1.2	History of TCP/IP protocols	1-1
1.2	Ethernet	1-2
1.2.1	Frame formats	1-2
1.2.2	Bus access method	1-3
1.2.3	Half duplex and full duplex operation	1-3
1.2.4	Auto negotiation	1-3
1.2.5	Ethernet and TCP/IP	1-4
1.3	Protocols and applications	1-5
1.3.1	Point-to-Point Protocol (PPP)	1-5
1.3.2	Internet Protocol (IP)	1-6
1.3.3	Internet Control Message Protocol (ICMP)	1-12
1.3.4	Transmission Control Protocol (TCP)	1-14
1.3.5	User Datagram Protocol (UDP)	1-19
1.3.6	OpenModbus on TCP/IP	1-21
1.3.7	BootP and DHCP	1-23
1.3.8	Address Resolution Protocol (ARP)	1-27
1.3.9	Other protocols and applications	1-28
1.4	Cabling	1-32
1.4.1	Network cables	1-32
1.4.2	Connector pin assignment	1-33
1.4.3	1:1 cables and crossover cables	1-34
1.4.4	Cable length restrictions	1-34
1.5	Network components	1-35
1.5.1	Terminal devices	1-35
1.5.2	Repeaters and hubs	1-36
1.5.3	Bridges, switches and switching hubs	1-39
1.5.4	Media converters	1-41
1.5.5	Routers	1-42
1.5.6	Gateways	1-42
2	The Ethernet coupler	2-1
2.1	Features	2-1
2.1.1	Supported protocols	2-1
2.1.2	Sockets	2-2
2.1.3	Restrictions	2-2
2.2	Technical data	2-2
2.2.1	Technical data of the coupler	2-2
2.2.2	Interfaces	2-2
2.2.3	Technical data of the Ethernet interfaces	2-2
2.3	Connection and transfer media	2-3

2.3.1	Attachment plug for Ethernet cable	2-3
2.3.2	Ethernet cable	2-3
2.4	Ethernet implementation	2-4
2.4.1	Configuration	2-4
2.4.2	Running operation	2-4
2.4.3	Error diagnosis	2-4
2.5	Diagnosis	2-5
3	Designing and planning a network	3-1
3.1	Introduction	3-1
3.2	Concepts for structuring a network	3-1
3.2.1	Hierarchy model	3-2
3.2.2	Redundant model	3-3
3.2.3	Safe models	3-4
3.3	Utilization and performance	3-5
4	Design examples	4-1
4.1	Introduction	4-1
4.2	General procedure for configuring the coupler	4-1
4.3	Ethernet used for online access	4-5
4.3.1	Configuring the coupler	4-5
4.4	Modbus on TCP/IP	4-8
4.4.1	Operation as server / slave	4-8
4.4.2	Operation as client / master	4-12
4.5	Fast data communication via UDP/IP	4-18
4.5.1	Example configuration for data communication via UDP/IP	4-18
4.5.2	Configuring the Ethernet couplers for data communication via UDP/IP	4-19
4.5.3	Implementation in the user program	4-20
5	Used terms and explanations	5-1
5.1	Used terms	5-1
5.2	Explanations	5-3
6	Figures	6-1
7	Tables	7-1
8	Index	I

1.1 History

Ethernet is the most commonly used network technology for many fields of application. Based on the physics of Ethernet, numerous protocols are used today. One of the best-known representatives of these protocols is TCP/IP.

1.1.1 History of Ethernet

The history of today's Ethernet goes back to the early seventies. At that time, Ethernet was originally developed by Xerox Corporation to link a printer. Then, on 13 December 1979, Xerox Corporation had Ethernet patented as 'Multipoint Data Communication System with Collision Detection' in the US Register of Patents under the number 4 063 220.

As a result, the DIX group, a consortium consisting of DEC, Intel and Xerox, started its further development. The goal of this consortium was to establish the Ethernet technology as a LAN standard. For this purpose, the company-internal specification of the Ethernet LAN (which is today called Ethernet version 1) was published in September 1980.

Later, the American standardization Institute IEEE founded a study group named 802.3 to work out an internationally accepted standard on the basis of this publication. On 23 June 1983, Ethernet was approved for the first time as standard IEEE 802.3. 1990 the 10BaseT standard IEEE 802.3i followed which specified the use of an unshielded twisted pair cable instead of the coaxial cable used before. Later, the original transfer rate of 10 Mbit/s was increased step by step and various other transfer media, e.g. optical fibres, were adapted with further standards.

1.1.2 History of TCP/IP protocols

The development of today's TCP/IP protocols (Transmission Control Protocol / Internet Protocol) originated in military and occurred independent of the development of the Ethernet. As early as the beginning sixties, the RAND corporation in USA accepted the challenge to develop a communication network which should be resistant against military attacks. Among other things, this concept included that the system had to deal without a susceptible central control. However, during the sixties this concept could not really make progress.

The actual history of TCP/IP protocols started in 1968, as a department of the American ministry of defense started a series of experiments for computer-to-computer connections based on multiplexed lines. In 1969 this resulted in the ARPANET project. The Advanced Research Projects Agency was the leading instance for the development of ARPAnet and at the same time provided the name. The ARPAnet enabled scientists to use computer data and programs on other people's computers located far away. In 1972, the ARPAnet already consisted of 37 nodes. The intensification of access control performed by the Pentagon resulted in a change of the name to DARPA net (D from defense).

At that time, the net structures (e.g. telecommunication via satellite, local area networks (LAN)) were completely new and required the further development of the previous NCP protocol. Thereupon, the ARPA study group 'Internet Working Group' (INWG) established principles for the data transfer between independent networks, which produced the Protocol-for-Packet Network Intercommunication. This resulted in the development of the Kahn-Cerf protocols which became well-known under the name TCP/IP protocols.

On 1 January 1983 the previous NCP protocol was finally replaced by the set of TCP/IP protocols. Likewise in 1983 the DARPA net, which was previously strictly kept under the control of the Pentagon, was divided into a military and non-military network.

1.2 Ethernet

1.2.1 Frame formats

One fundamental part of the Ethernet specification is the arrangement of the data transfer format. When transferring data via Ethernet, the actual user data are preceded by a so-called preamble (which is among other things used to synchronize the receiver stations) as well as the hardware source and target address and a type length field. A checksum follows after the user data. All information mentioned above together constitute an Ethernet frame. During the development of the Ethernet, different types of frames arose. The following figure shows the structure of an Ethernet 802.3 frame.

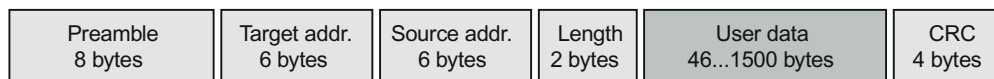


Figure 1-1: Structure of an Ethernet 802.3 frame

It has to be observed that the transferred user data do not inevitably contain only useful information. When transmitting data using a protocol above Ethernet (refer to section 1.2.5 "Ethernet and TCP/IP"), each protocol layer passed prior to the actual transmission supplements the original user data by its specific frame or header, so that the maximum number of actual user data is smaller, depending on the used protocols.

MAC address:

Each Ethernet terminal device that has the MAC layer functions implemented (refer to section 1.2.5 "Ethernet and TCP/IP") has a world-wide unique hardware and MAC address. Inside of this 6 bytes long address, the two most significant bits of the first byte have particular functions. The most significant bit is also called the I/G bit (Individual/Group bit) and indicates whether it is an individual world-wide unique address (unicast address, I/G bit = 0) or a group address (I/G bit = 1). The second most significant bit is called the G/L bit and indicates whether it is a globally or a locally administered MAC address. A GAA (Globally Administered Address, G/L bit = 0) is an address which is fixed programmed by the device manufacturer and has to be unique all over the world. An LAA (Locally Administered Address, G/L bit = 1) can be a MAC address which has been changed afterwards for the use within a network. For this, it has to be observed that a MAC address has to be unique within a network.

The first 3 bytes of a MAC address are the manufacturer-related address part. Using this value, the manufacturer of an Ethernet chip can be determined. Each manufacturer of Ethernet components has one or several pre-defined address ranges assigned he can use for his products. For example, 3COM uses among others the MAC address range 02-60-8C-xx-xx-xx.

For Ethernet, the MAC address is represented in a canonical form. This representation starts with the least significant bit (LSB) and ends with the highest significant bit (MSB) of a byte. The following figure shows a global unicast address of the manufacturer 3COM.

Canonical representation 02-60-8C-00-00-01

Binary representation 01000000-01100000-00110001-00000000-00000000-10000000

1.2.2 Bus access method

Ethernet uses the CSMA/CD access method (Carrier Sense Multiple Access / Collision Detection). With this method, the station that wants to transmit data first "listens" to the carrier whether data are already transmitted by another station (carrier sense). If the carrier is busy, the station later tries to access the carrier again. If the carrier is idle, the station starts the transmission.

With this method, particularly in greater networks it can happen that several stations try to transmit at the same time (multiple access). As a result, they "listen" to the carrier, detect that the carrier is free and correspondingly start the transmission. This can cause collisions between the different data packages. This is why each station verifies whether a collision occurred during transmission (collision detection). If this is the case, the station aborts the transmission and then tries to send its data again after a wait time which is determined by a random generator.

Collisions within an Ethernet network do not cause loss of data, but they reduce the available bandwidth of the network. In practice, for a network with 30 stations on the bus, a net bandwidth of approx. 40 % is assumed. This means that a bandwidth of only approx. 4 Mbit/s is available in a network with a theoretical bandwidth of 10 Mbit/s, for instance. This has to be considered when planning an Ethernet network. The number of collisions can be reduced to a minimum if the network is carefully planned and if only suitable network components are used (refer to section 1.4 "Cabling" and 1.5.4 "Media converters").

1.2.3 Half duplex and full duplex operation

If communication is only possible in one direction (transmission or reception), it is called half duplex mode. However, the separate transmit and receive lines of today's twisted pair cabling for Ethernet networks also allows full duplex operation. In full duplex mode, the stations can simultaneously exchange data in both directions independent from each other. Due to this, the CSMA/CD method is not necessary in full duplex mode. Networks with more than two stations working in full duplex mode can only be implemented using switches because these switches establish peer-to-peer connections between the individual stations (refer to section 1.4 "Cabling").

1.2.4 Auto negotiation

Today, Ethernet uses transmission rates of 10, 100 or 1000 Mbit/s in half duplex or in full duplex mode. However, not all devices support all possible settings. This particularly makes the optimum network configuration more difficult for networks using twisted pair cables of the same kind and components which can be used with 10 Mbit/s or 100 Mbit/s in half duplex or in full duplex mode as desired. Imperfect configurations can lead to link errors or at least to performance losses because the maximum possible transmission rate is not used.

Due to this, the auto negotiation functionality (in the past also called Nway) has been established with the introduction of Fast Ethernet. With this functionality the stations agree on the highest possible transmission rate and, if possible, full duplex operation. Then, all subscribers on the network configure themselves optimally.

However, problems could arise if one component in one segment is configured manually, i.e. if it has been set to a fixed transmission rate and mode and the auto negotiation function has been switched off. In this case, a device operating in auto negotiation mode informs the manually configured device about its possible settings but does not receive any response.

1.2.5 Ethernet and TCP/IP

Like nearly all standards in the field of data transmission, Ethernet also follows the ISO/OSI layer model. Based on this reference model the principle course of a transmission is described. Each of the 7 parts (layers) has a particular function and makes it available for the next higher layer.

The Ethernet standard IEEE-802.3 defines the function of the two lowest layers. These layers consist of the following components and the Logical Link Control (LLC) which is described in the IEEE standard 802.2.

- Media Access Control Protocol (MAC)
- Physical Layer Signaling (PLS)
- Attachment Unit Interface (AUI)
- Medium Dependent Interface (MDI)
- Physical Medium Attachment (PMA)

Since the ISO/OSI model did not yet exist when the development of TCP/IP protocols started, these protocols are based on the DoD architecture. The DoD model cannot be clearly transferred to the ISO/OSI model. The following figure shows a comparison of Ethernet in the ISO/OSI model and the TCP/IP protocols adapted to that model. This shall explain that Ethernet does not necessarily mean TCP/IP (and vice versa). To be precise, TCP/IP is only based on Ethernet and can also be used in other data networks (e.g. for satellite links). In return, in Ethernet networks not exclusively the TCP/IP protocol is used. Actually, TCP/IP is only one of numerous protocols which are used side by side.

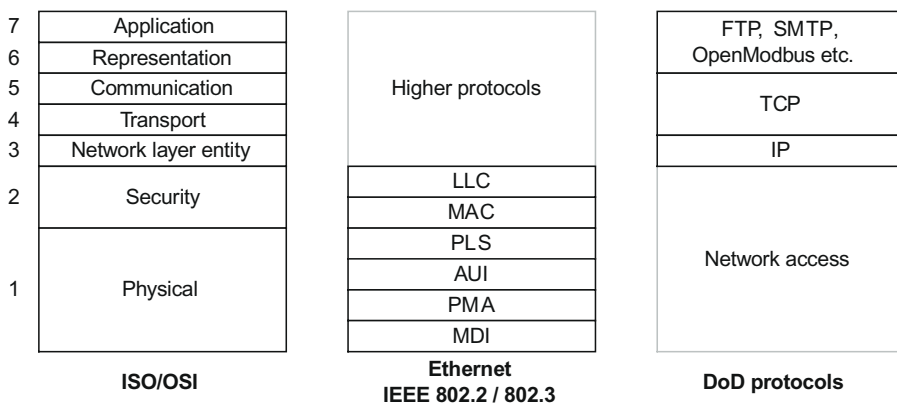


Figure 1-2: Ethernet in the ISO/OSI model

1.3 Protocols and applications

Section 1.2.5 "Ethernet and TCP/IP" only describes the context between Ethernet and TCP/IP. But actually numerous protocols are used in parallel in Ethernet networks. Furthermore, the protocols described here are not tied to Ethernet, as shown in the following figure.

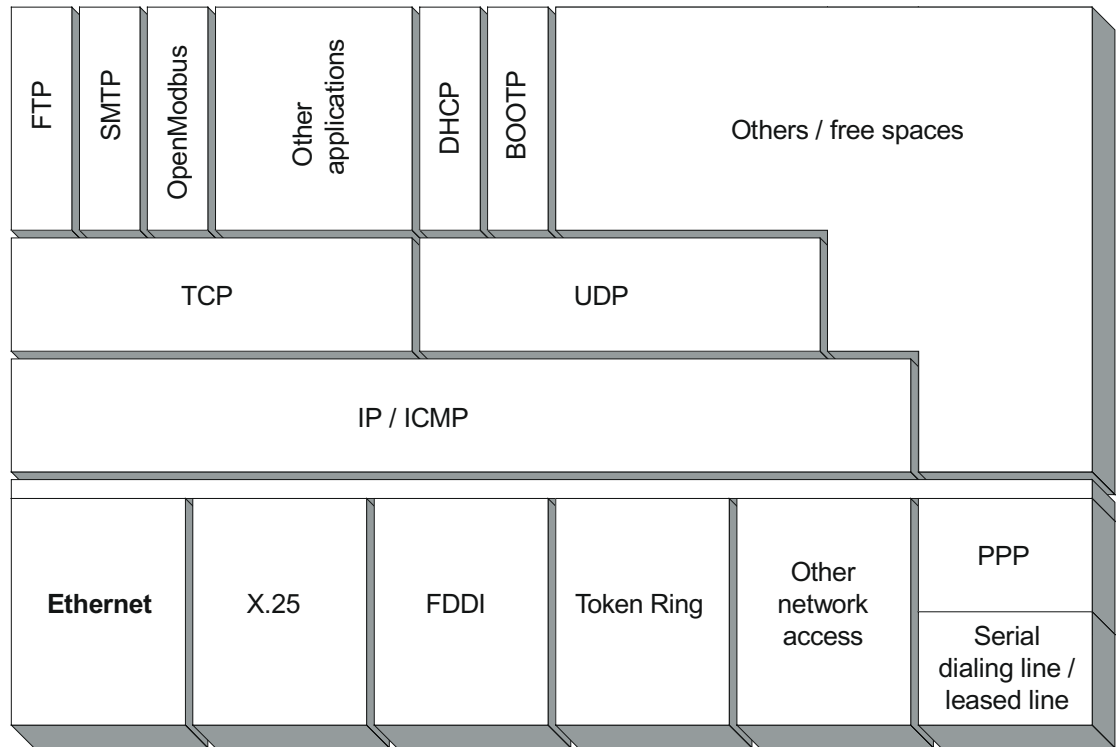


Figure 1-3: Ethernet protocols

1.3.1 Point-to-Point Protocol (PPP)

PPP is a protocol for the transmission of LAN protocols (e.g. IP) via wide area networks. Thus, the point-to-point protocol enables the transmission of data via synchronous and asynchronous dial and leased lines. Because PPP is working independent of the corresponding physical interface, it can be used for modem connections as well as for ISDN connections or on RS232.

1.3.2 Internet Protocol (IP)

With the TCP/IP protocols, the Internet protocol takes on the tasks of the network layer. IP transmits the data packages of the higher protocol layers (incl. their headers) in the form of so-called datagrams via the network. The transport mechanism used for IP is connectionless, which means that the IP module of a transmitter does not verify whether the recipient actually received the data. This task has to be performed by higher protocol layers (e.g. TCP). Through the internet protocol, each datagram is transmitted as a single individual data package. During the transmission, an independent route of transport is determined for each datagram in a network. With this, it can occur that IP datagrams pass themselves on their way to the recipient and correspondingly arrive in a changed order. The task to sort the packages into the correct order is left to the protocols on the transport layer.

If a protocol of a higher layer hands over data to be transmitted to the IP protocol, the IP protocol first generates a datagram consisting of the data to be handed over and the IP header. If the target station is located in the local network, IP directly transmits the datagram to this station. If the target is located in a remote network, the datagram is first transmitted to a router inside the local network which forwards the datagram to the target station, if necessary via further routers. During this process, each router involved in the transport of the datagram generates a new IP header.

Because in today's complex networks often several routes of transport are available between the sender and the recipient, several subsequent telegrams are not necessarily routed via the same way. Furthermore it can occur during the transport between the sender and the recipient that data packages are routed via networks with a maximum permitted telegram length smaller than the length of the datagram to be transmitted. In this case the adjacent router disassembles the datagram to smaller data packages. This process is referred to as fragmentation. The individual fragments are provided with an own IP header and forwarded as independent data packages which can be routed to the target station via different ways and therefore also in a different order. The target station then has to assemble the data fragments to one single datagram before this datagram can be handed over to the next higher layer. This process is called re-assembly mechanism.

IP header:

Each datagram transmitted via IP has a preceding IP header. Among other things, this header contains the IP addresses of the sender and the recipient, a checksum and a time to live identifier. The following figure shows the principle structure of an IP header.

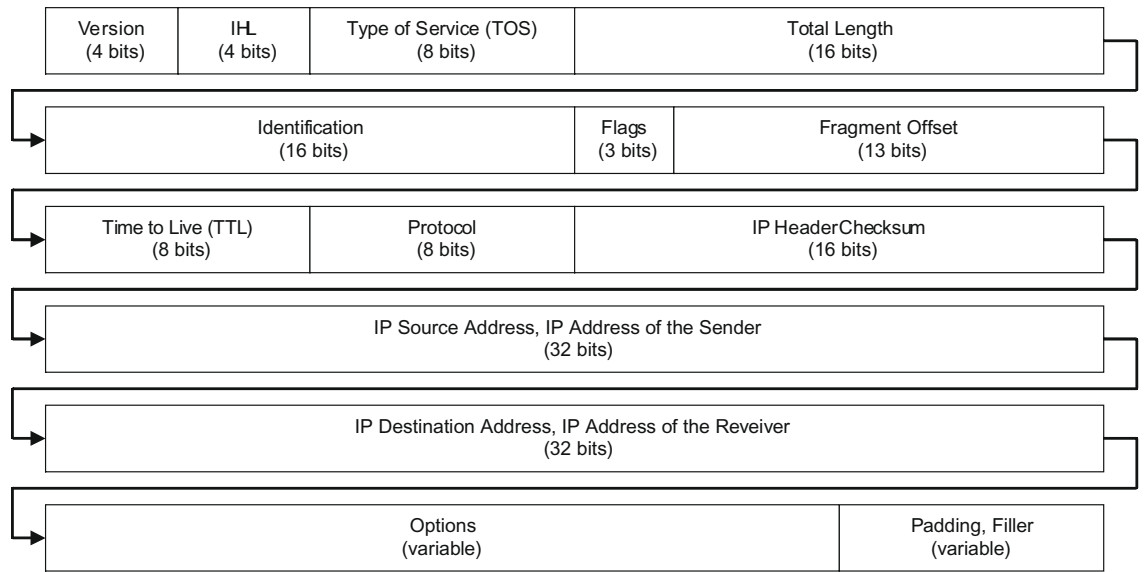


Figure 1-4: Principle structure of an IP header

The total length of an IP header is defined in the IHL field (Internet Header Length) in numbers of 32 bit units (4 octets) and is, due to the variable size of the options field, 20 bytes or longer (5 x 4 octets = 20 bytes).

The following figure explains that an IP datagram embedded in the Ethernet frame reduces the actually transmittable user information. Further reductions result from the headers of the higher layers.



Figure 1-5: IP datagram in the Ethernet frame

Detailed descriptions of the content and meaning of the individual fields of the IP header are not given here, since such detailed knowledge is not necessary for normal application. Only the meaning of the checksum and the time to live identifier (TTL) as well as the structure of the IP addresses are described in the following sections.

The checksum is used by the recipient of an IP datagram to verify that the data package has been transmitted without errors. Datagrams with a faulty checksum are dismissed by the routers involved in the routing of the datagram as well as by the final recipient.

Each time a router receives an IP datagram, it reduces the time-to-live timer of the datagram by 1. If the value reaches 0, the datagram is destroyed or no longer forwarded. This shall prevent datagrams from endless straying in the network.

IP addresses:

Each station using the IP protocol must have at least one IP address. The IP address must not be confused with the Ethernet MAC address (refer to section 1.2.1 "Frame formats"). An IP address is a 32 bit value which is normally represented using the 'dotted decimal notation' (e.g. 89.129.197.1).

An IP address is divided into two parts: the network address and the computer's address (node address). The network address part includes up to three bytes. The size of the network address which determines the address class, is defined by the first four bits of the first octet of the IP address. The following table gives an overview of the various classes with their identification and use.

Class	1. octet	Use
Class A	0 – 127	Few networks, many computers
Class B	128 – 191	Medium distribution of networks and computers
Class C	192 – 223	Many networks, few computers
Class D	224 – 239	Multicast addresses
Class E	240 – 255	not defined

Table 1-1: Classes of IP addresses

Class A addresses consist of one octet for the network address and three octets for the computer address. For example, the IP address 89.129.197.1 designates computer 129.197.1 in network 89. Since the highest bit of the network address is always 0 and the IP addresses 0 and 127 are of particular significance, only 126 networks can be addressed by a class A address. Using the three octets left for the computer address, up to 16.777.216 computers can be addressed within these networks. Value 255 is reserved in each octet of the IP address for the broadcast address.

Class B addresses consist of two octets for the network address and two octets for the computer address. For example, the IP address 129.89.197.1 designates computer 197.1 in network 129.89. Since the two highest bits of the network address always have a value of 10, the address range reaches from 128 to 191. The second octet is also interpreted as a part of the network address. This way, 16.384 networks with up to 65.536 nodes each can be addressed with the class B address. Value 255 is reserved in each octet of the IP address for the broadcast address.

Class C addresses consist of three octets for the network address and one octet for the computer address. For example, the IP address 197.129.89.1 designates computer 1 in network 197.129.89. The three highest bits of the network address always have a value of 110. Consequently, the values 192 to 223 are valid in the first octet and the values 1 to 254 in the subsequent two octets. This results in an address range of 2.097.152 networks with up to 254 nodes each. Value 255 is reserved in each octet of the IP address for the broadcast address.

The range 224 to 239 (class D) is intended for multicast addressing as it is frequently used today for newer protocols of the IP protocol family.

Class E addresses are reserved for future use.

The following table gives an overview of the IP address structure for the individual classes.

Network address	1. octet							2. octet							3. octet							4. octet														
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7				
Class A	0	Network address							Computer address																											
Class B	1	0	Network address														Computer address																			
Class C	1	1	0	Network address																								Computer address								
Class D	1	1	1	0	Multicast address																															
Class E	1	1	1	1	undefined format																															

Table 1-2: Overview of IP address structures

IP address assignment:

The allocation of IP addresses or the assignment of an IP address to a device can be done in different ways (see also section 2 "The Ethernet coupler").

The simplest case is the static assignment of the IP address. Here, the device is configured with a fixed address which is assigned by the network administrator. Since the IP address always includes the identification of the network the device is used in, the network administrator has to be informed about the device's place of installation. One disadvantage of this method appears, if the device shall be used in another network later. In this case, an IP address has again to be requested from the administrator and set manually. The advantage of this method is that the IP address of the device is known and that the device can always be called under this address.

Alternatively the IP address can also be obtained from a BOOTP or a DHCP server (refer to section 1.3.7 "BootP and DHCP"). In this case the IP address is assigned on the basis of the device's MAC address when switching on or booting the device. Here we have to distinguish static and dynamic assignment. The advantage of both methods is that no further settings have to be performed for the device. This also applies if the device is later used in another network. The disadvantage of the dynamic method is that the IP address of a device can possibly change when it is switched on again the next time. Therefore, it is not possible to implement a fixed communication connection to another device (e.g. two controllers via function blocks).

For all methods mentioned above, fixed implemented connections to other devices have to be implemented again if the IP address of the device has changed. In the first two cases (direct address configuration on the device and automatic static assignment) this will normally not be necessary since the IP address of the device has only to be changed if the device shall be used in another network. In the latter case (dynamic IP address assignment) this problem is not only restricted to inter-network connections since the IP address of a device can change each time the device is switched on, even if both devices involved in the connection are located within the same network or if their places of installation do not change.

Special IP addresses:

As already mentioned in the section before, special IP addresses with a special function exist. These addresses are concisely described below.

127.x.x.x

The class A network address with the number 127 is reserved for the loopback function of a station, independent of its network class. By definition, all IP addresses beginning with 127 may only be used for internal testing. They must not be transmitted via the network. Datagrams addressed to network address 127 are returned to the sender within the TCP/IP software (loop). This allows testing of the software function. The network controller itself is not tested this way. If the network controller shall be included in the test, it is recommended to use the ping application.

Value 255 in an octet

The value 255 is reserved as a broadcast address (all one broadcast). To address all computers within the same network, the computer's part of the network address has to be set to 255 (e.g. 91.49.1.255). The use of subnet masks possibly complicates the calculation of the broadcast address since in this case an extended algorithm has to be used.

Value 0 in an octet

There are two definitions for the use of the value 0 within an IP address. An earlier definition intended value 0 as a broadcast. This all zero broadcast should no longer be used today. Now, the all one broadcast (see above) should be used instead. The second definition for the address value 0 which is presently valid allows the identification of the own network (this net) or computer (this host). For example, an IP address of 0.1.1.1 means that computer 1.1.1 in this network shall be called.

Subnet mask:

When configuring a device which is connected to a TCP/IP network it is often possible not only to set the IP address but also to preset a so-called subnet mask. If the IP address of the device is configured manually, the subnet mask has also to be entered manually, if it is used. If the device is set to automatic IP address assignment (refer to section "IP address assignment" and section 1.3.7 "BootP and DHCP"), the assignment of a possibly required subnet mask is also performed automatically.

The use of subnet masks allows network administrators to divide a large network into several small (sub) networks. Subnetworks are often used to image the topological structure of large networks (e.g. the departments or floors within one building) or to decentralize the administration of the network. Furthermore, logical transitions between different transfer media can be easily implemented this way. An IP router is able to connect different physical networks to each other. The only condition for this function is the existence of an own network address for the router in each connected network. A subnet is only known by the computers connected to the local network. Outside of the local network, such an address appears as a usual IP address.

With this method which is called 'subnetting', specific bits of the computer's IP address part are used to expand the class-specific network address part. This reduces the address range available for the addressing of computers in the subnet. The IP address is then divided into the network address, the subnet number and the computer's address. The subnet number is defined by means of a bit mask (subnet mask). With this, the IP address and the subnet mask

are bit-wise logically interconnected by an AND. Consequently, the subnet mask defines which parts of an octet are to be interpreted as the (sub) network or as the node address. As a result, the software of the IP protocol distinguishes devices connected to the local subnet and device located in other subnets. The computer is located in the local network if the result of the AND interconnection of the IP address and the subnet mask delivers the local network address and the local subnet number. Otherwise the datagram is routed to another subnet router.

For example, by applying a subnet mask of 255.255.0.0, a class A address (e.g. 126.x.y.z) with a default subnet mask of 255.0.0.0 becomes a class B address and by applying a subnet mask of 255.255.255.0 this address becomes a class C address. The use of subnet masks is explained in the following examples.

Example 1:

Computer A with the IP address 89.236.4.85 and the subnet mask 255.224.0.0 wants to establish a connection to computer B with the IP address 89.234.85.50.

IP address computer B (bin)	01011001	11101010	01010101	00110010
Subnet mask computer A (bin)	11111111	11100000	00000000	00000000
AND interconnection (bin)	01011001	11100000	00000000	00000000

The result corresponds to the IP address of computer A expanded by the subnet mask. Consequently, computer B is located in the local subnet and can be addressed directly.

IP address computer A (bin)	01011001	11101100	00000100	01010101
Subnet mask computer A (bin)	11111111	11100000	00000000	00000000
AND interconnection (bin)	01011001	11100000	00000000	00000000

Example 2:

Computer A with the IP address 89.236.4.85 and the subnet mask 255.224.0.0 wants to establish a connection to computer C with the IP address 89.211.1.22.

IP address computer C (bin)	01011001	11010011	00000001	00010110
Subnet mask computer A (bin)	11111111	11100000	00000000	00000000
AND interconnection (bin)	01011001	11000000	00000000	00000000

The result does not correspond to the IP address of computer A expanded by the subnet mask (refer to example 1). Consequently, computer C is not located in the local subnet. The datagram has to be transferred to computer C via a router.

Example 3:

A network with the class A address 126.x.y.z can be divided into two subnets by means of the subnet mask 255.128.0.0 where one subnet covers the range from 126.0.y.z to 126.127.y.z and one the IP address range from 126.128.y.z to 126.255.y.z. For the broadcast addresses of the subnet masks the addresses 126.127.255.255 and 126.255.255.255 respectively are automatically used since in this cases all bits of the host part of the address are set to 1 (all one broadcast).

Gateway:

Another parameter that can be set in the IP software of some devices is the gateway or standard gateway parameter. This parameter defines the IP address of the gateway (router) to which datagrams addressed to a computer outside of the local network are to be transferred. If the IP address of the device is configured manually, the IP address of the gateway has also to be entered manually, if it is used. If the device is set to automatic IP address assignment (refer to section "IP address assignment" and section 1.3.7 "BootP and DHCP"), the assignment of a possibly existing gateway is also performed automatically.

1.3.3 Internet Control Message Protocol (ICMP)

Aside from the main protocol IP the auxiliary protocol ICMP also exists on the network layer. ICMP is used to exchange information and error messages between the network subscribers. The ICMP protocol is based on the IP protocol and is thus treated by IP like a protocol of a higher layer. ICMP data are always transferred together with a complete IP header. The actual ICMP messages are included in the subsequent IP data part.

The following **information messages** are defined in the ICMP protocol:

- Echo
- Information
- Address mask / address format
- Router discovery

Echo request/reply message

The recipient of an echo request datagram returns all data contained in the received data package to the sender by means of an echo reply datagram.

The best-known tool based on ICMP echo packages is the ping command. Ping (Packet Internet Gopher) is implemented in virtually all IP terminal devices. It is based on the principle that one device sends an ICMP echo request and then waits for the response. Depending on the implementation only a success message (host alive) or a failure message (no response from host) is output or even the wait time for the response.

Information request/reply message

By means of an information request message the sender of a datagram is able to poll the network address of the network it is connected to. For this purpose, it sends a corresponding request to IP address 0 and then receives a reply message with the local network address from any device connected to the network.

Timestamp request/reply message

In a timestamp request the sender transfers its local time as the sending time in the datagram to the recipient. The recipient of the request then repeats the sender's sending time in its reply message to the sender and supplements this by its local reception time of the request and the local sending time of the reply.

Address format request/reply message

By means of an address format message the sender of a datagram is able to poll the subnet mask length or address in bits from another subscriber in the network.

Router discovery message

By means of the router discovery messages a device in the network is enabled to automatically determine the IP addresses of the connected routers (gateways). It is distinguished between router advertisement messages and router solicitation messages. With router advertisement messages a router periodically transmits the availability of its network interfaces by multicast. In contrast to this, router solicitation messages are sent by a network node to one or more routers after its initialization. The routers that received this message then return one single router advertisement message.

Furthermore, the following ICMP **error messages** exist:

- Destination unreachable
- Redirect
- Source quench (resources exhausted)
- Time exceeded
- Parameters problem

Using the ICMP error messages the sender of a datagram is informed about the reason why the datagram could not be transmitted.

Destination unreachable message

A destination unreachable message can have different causes. For example, it is possible that the higher protocol on the target computer is unknown or that the called destination port is already in use. Interruptions of the network can also cause destination unreachable messages.

Redirect message

Using a redirect message the sender of a datagram is informed via which router the datagram can be transmitted to the target network. This message is generated by a router if it receives a datagram which is addressed to a target network that cannot be reached via this router. However, the router knows the IP address of the correct router and informs the sender about this address using a redirect message.

Source quench message

A source quench message is for example output by a target computer if it is not able to process the amount of received data fast enough. Using this message the sender is requested to reduce the data transfer rate until no more source quench messages are received.

Time exceeded message

One reason for a time exceeded message could be for example that the IP protocol is not able to re-assemble a fragmented datagram within a defined period of time to a full data stream during the re-assembly process.

Parameter problem message

For example, a parameter problem message is issued if a router detects an error in the IP header of a datagram (e.g. an incorrectly set argument in the IP options) and consequently destroys the datagram or doesn't forward it.

1.3.4 Transmission Control Protocol (TCP)

The transmission control protocol implements the tasks of the transport layer in the ISO/OSI model and therefore is directly based on the IP protocol. TCP works connection-oriented. It makes connection services available for an order-accurate and secure transmission of user data. TCP enables the detection of data losses, the automatic re-transmission of lost data packages as well as the detection of duplicates. Compared with TCP, connectionless operating protocols of the transport layer which do not support such mechanisms (e.g. UDP) provide higher performance.

TCP treats the data handed over by the higher protocols as one quasi-continuous data stream, segments these data and transmits them in single data packages. On the receiver's side the single segments are re-assembled again and made available for the higher protocols as a data stream. The TCP protocol is absolutely independent of the individual network-specific properties. Hence, a TCP segment can have a length of up to 65 kbytes. When data are transmitted via a network, the lower IP protocol has to fragment the TCP segments further into several small packages of a low order according to the properties of the network (e.g. Ethernet). The following figure shows the course of the transfer processes.

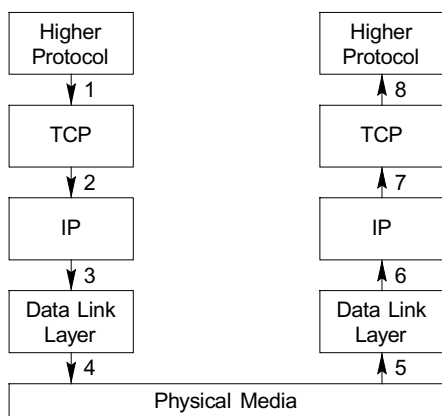


Figure 1-6: Course of the transfer processes

Port:

The multiplex mechanism of the transmission control protocol (TCP) allows the simultaneous use of TCP by a large number of higher protocols and application processes. The assignment of the individual data streams to the above applications is done via so-called ports. For the access to specific standard applications firmly defined port numbers exist (well-known ports). These port numbers range from 0 to 1023 and are assigned by the IANA (Internet Assigned Numbers Authority). All other ports from 1024 to 65535 are not under the control of the IANA and can be used as desired.

In the following table some well-known ports are listed.

Application	Port number
FTP data (FTP data channel)	20
FTP control (FTP control channel)	21
Telnet	23
SMTP (Simple Mail Transfer Protocol)	25
DNS (Domain Name Server)	53
http (HyperText Transfer Protocol)	80
OpenModbus	502

Table 1-3: Well-known ports

Socket:

For unique identification of a TCP application process, the port number and the IP address of a device are locally summarized to a so-called socket. A socket with a firmly defined port number is designated as well-known port. The following simplified example explains the principle of the logical connections between pairs of sockets.

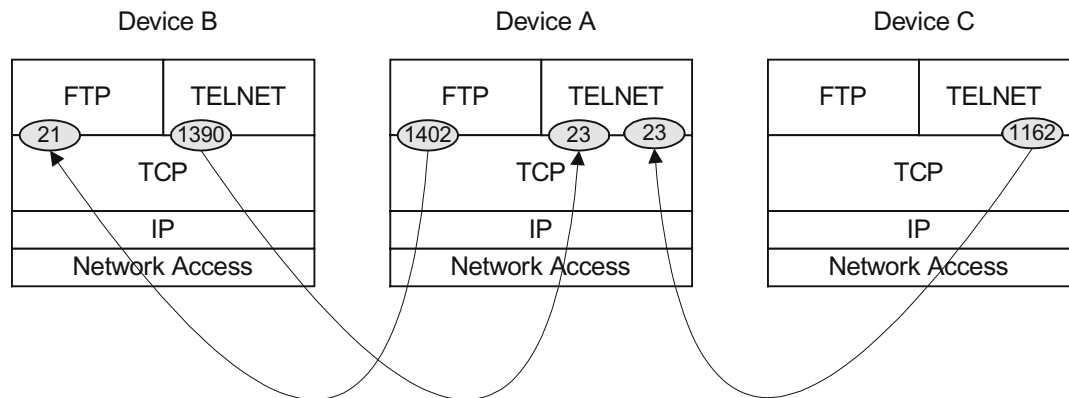


Figure 1-7: Principle of the logical connections between pairs of sockets

The devices A, B and C are located within the same network. Each device has a unique IP address assigned. The individual TCP modules have several independent ports with unique port numbers.

Device A wants to exchange files with device B via FTP and establishes for this purpose a connection from its local port 1402 to the well-known socket 21 of device B. For this, the local port number of the device that initiated the connection (here: device A) is not of importance because the corresponding application is already uniquely defined by the active connection establishment. The connection establishment only requires that the request can be identified as FTP control in device B through the use of the socket or port 21.

At the same time device B dials in (from local port 1390) to device A. There it uses port 23 (Telnet). Since it's a telnet access, the well-known socket 23 is several times available in device A. Therefore, device C is able to establish a parallel telnet connection to device A via its local port 1162.

After the respective logical connection has been established successfully, all application data of the respective active device to be transferred are disassembled into TCP segments and transferred to the corresponding port of the respective remote device. Each port of the TCP module of a remote device accepts and sends only data from and to the TCP module of the

corresponding active device. In our example, device A has only two telnet ports. This is why every request for another connection to this port is denied as long as both connections are open. Once all data have been exchanged within the bounds of a connection, the respective connection is closed and the corresponding port is available again for new requests.

TCP header:

Each data package transmitted via TCP has a preceding TCP header. Among other things this header contains the respective ports of the sender and the recipient. The meaning of the term ports is explained earlier. Detailed descriptions of the further components of the TCP header and their functions are not given here, since such detailed knowledge is not necessary for normal application. The following figure shows the principle structure of a TCP header.

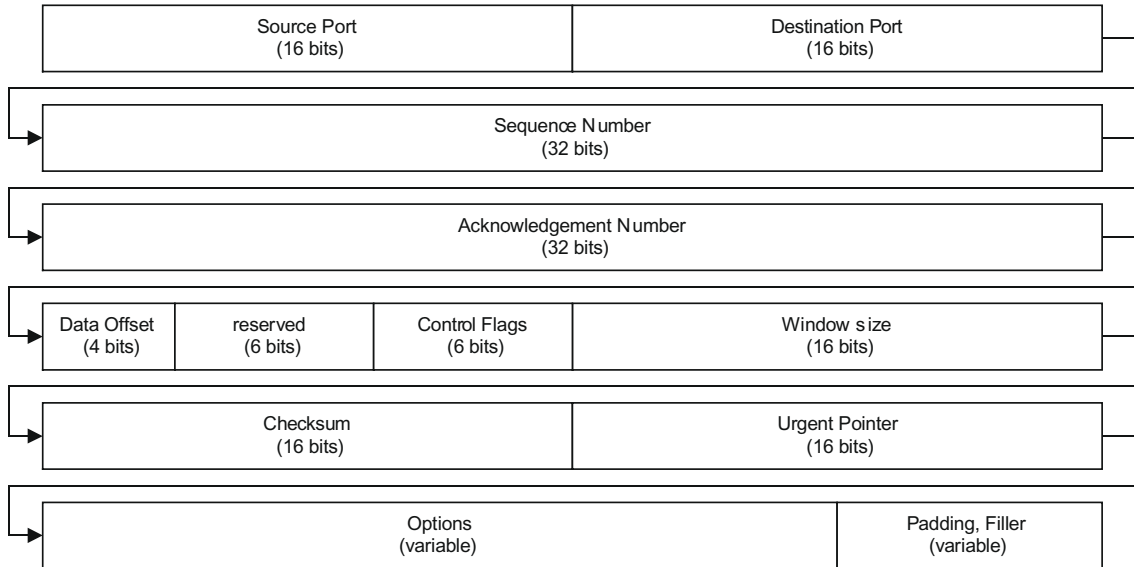


Figure 1-8: Principle structure of a TCP header

The total length of the TCP header depends on the type and number of options and is at least 24 bytes.

The following figure explains that a TCP segment embedded in an IP datagram and then packed in the Ethernet frame reduces the actually transmittable user information. Further reductions result from the headers of the higher layers.

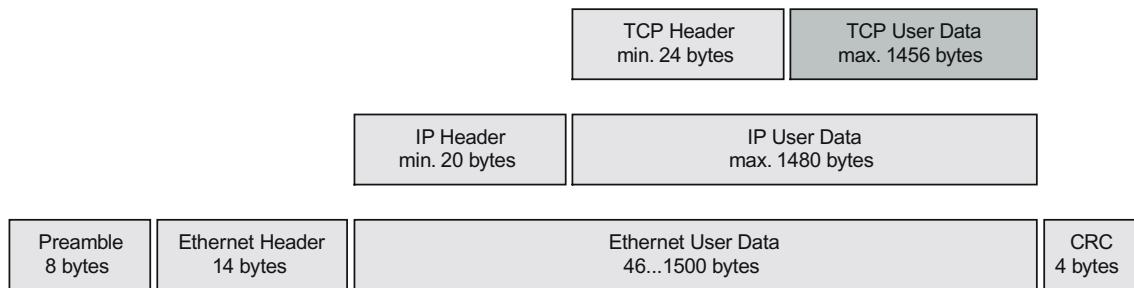


Figure 1-9: TCP segment in the Ethernet frame

Phases of a communication:

During the previous sections it was only explained that TCP connections first are established, then guarantee a safe transmission and finally are closed again after the transmission is finished. For a better understanding of these processes in the TCP protocol the context is explained in more detail below. The following figure shows a simplified state diagram of a TCP module.

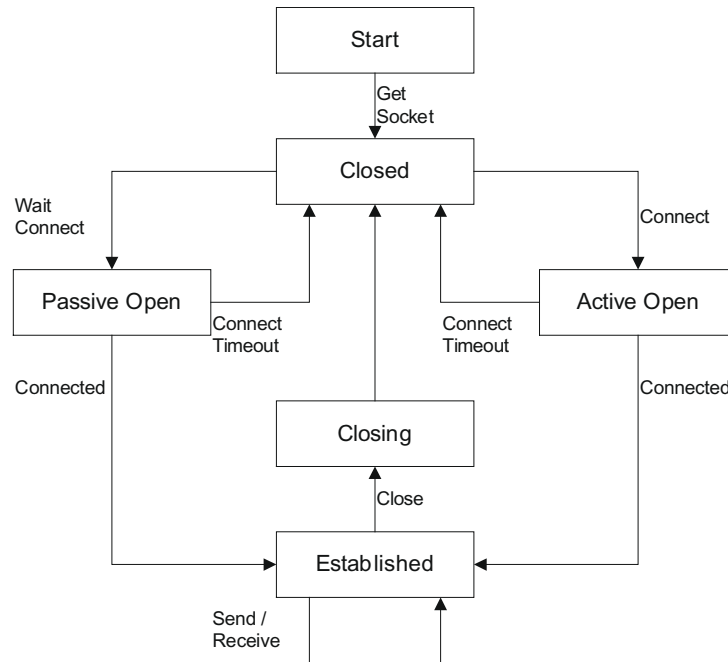


Figure 1-10: Simplified state diagram of a TCP module

In the figure above it is assumed that a device is switched on or that an application is started (state: start). The first TCP action starts with the request of a socket. After a successful assignment of a socket through the TCP module, the actual TCP starting point is reached. Starting from this state there are two possibilities to open a connection, in active mode or in passive mode. In active mode an active connection establishment to a network node is tried (Connect) whereas a device in passive mode waits for an incoming connection request (Wait Connect). Between the active and the passive device first various connection parameters are negotiated. Then the connection is established (Established) and data can be exchanged between the higher protocols (Send / Receive). After all data have been exchanged the disconnection (Close) is initiated. After a further exchange of connection parameters (Closing) the connection is closed and the TCP module is again set to the Closed state.

Aside from different safety mechanisms, each TCP state has **timing supervision functions**. Since some of these timers can be parameterized by the application or by the user, we want to briefly explain their functions below.

Connect Timeout

The connect timeout function determines the time period after which an attempt of a connection establishment shall be aborted. In active mode, setting this time period is obligatory, i.e. a finite time value has to be preset for the connection setup. If the TCP module cannot establish the specified connection within this time, the attempt is aborted with an error message. Possible reasons for this could be for example that the addressed node is not located inside the network or that no corresponding socket is in the Wait Connect / Passive Open state.

In passive mode presetting of a connect timeout value is optional. If a time period is specified, the list state (Passive Open) is left with an error message if no attempt for a connection setup to the local socket occurred during this period. If no time period is specified, TCP waits for an incoming connection for an unlimited time.

Retransmission Timeout

Following a successful connection establishment the TCP data transmission phase takes place. The reception of a data package is confirmed to the sender by a positive acknowledgement (ACK) of the recipient. Due to the flow rate only one single ACK message is used to confirm the reception of a whole series of packages instead of confirming each single data package. The confirmations of reception are monitored on the sender's side using the so-called retransmission timer. One retransmission timer is started each time a data block is sent. If (e.g. due to a transmission error) no confirmation is received before the timer has elapsed, the package is sent again.

Give-Up Timeout / Send Timeout

The give-up timer is also used for the timing supervision of the reception confirmations for data segments sent by the recipient. After this time period the connection is closed if no data segments have been confirmed by the target node.

Inactive Timeout

The inactive timeout is used to monitor the activity on a connection and is normally implemented by the application. The timer is newly started with each data reception. If no data packages are received before this timer is over, the connection is automatically closed. The use of the inactive timeout function is optional. The inactive timeout cannot be used together with the keep alive mechanism (described below).

Keep Alive Timeout

The keep alive mechanism can be used to test whether a connection that did not carry data over a longer period of time still exists. The timer is newly started with each data reception. After this period of time the keep alive timer generates data packages (so-called keep alive probes) which are sent to the other side of the connection. If these packages are not confirmed, the connection is closed. The use of the keep alive timeout function is optional. The keep alive mechanism cannot be used together with the inactive timeout (described above).

Close Timeout

A TCP connection is closed after the data transmission is finished. There are two possible methods for closing a connection, a graceful close or an abortion. Graceful close describes the normal and coordinated closing of a connection. Both communication partners come to an agreement about closing the connection. Optionally, this phase can be monitored by defining a close timeout value. The respective timer is started with the request for a connection establishment. If the connection cannot be closed in a coordinated manner within the specified time period, the connection is aborted (hard close). A close timeout value of 0 results in an immediate abortion of the connection. An abortion represents the closing of a connection forced by one side. This can lead to a loss of all data on the communication connection.

1.3.5 User Datagram Protocol (UDP)

In addition to TCP the user datagram protocol is a second protocol implemented on the transport layer (layer 4). It is also directly based on the IP protocol. In contrast to TCP, UDP is operating connectionless and not connection-oriented. It does not provide end-to-end control. Transmission via UDP does not guarantee the correct delivery of a datagram to the recipient, the detection of duplicated datagrams and the order-accurate transmission of the data. If necessary, this has to be guaranteed by suitable mechanisms on the application side.

Compared to TCP, UDP has the advantage of a considerably higher performance. This is particularly reached by the missing safety mechanisms. No logical connections are established. The individual datagrams are treated as completely independent events instead. This particularly becomes apparent for multicasts or broadcasts. In case of TCP, first numerous connections have to be established for this, then the data have to be transmitted and finally the connections have to be closed again. For UDP this is restricted to the pure data transmission. In the end, the functions mentioned before result in the fact that the UDP header is considerably shorter than the TCP header and thus causes less protocol overhead.

Port:

Like TCP, UDP provides several ports (access addresses) to allow simultaneous access to the protocol for several processes (see also "Port" in section 1.3.4 "Transmission Control Protocol (TCP)"). The assignment of the port numbers to the application processes is performed dynamically and optional. However, for specific frequently used standard processes fixed port numbers (assigned numbers) are defined. The following table shows some of these assigned numbers.

Application	Port number
<i>Reserved</i>	0
Echo	7
IEN 116	42
Domain Name Service	53
BootP	67/68

Table 1-4: Fixed port numbers (assigned numbers) for standard processes

Socket:

For unique identification of a UDP application process, the port number and the IP address of a device are locally summarized to a so-called socket. A socket with a firmly defined port number is designated as well-known port. A pair of sockets uniquely identifies a pair of processes communicating with each other (see also the example under "Socket" in section 1.3.4 "Transmission Control Protocol (TCP)").

UDP header:

Each data package transmitted via UDP has a preceding UDP header. This header is considerably shorter and simpler than the TCP header and only contains the source and the destination codes (port number of the source in the sender and the target in the recipient) as well as the total length of the data package and a checksum. The UDP user data immediately follow the header. The following figure shows the principle structure of a UDP header.

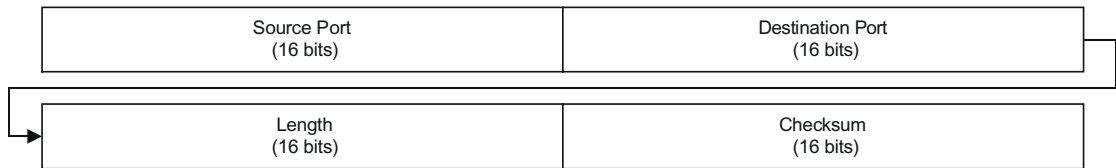


Figure 1-11: Principle structure of a UDP header

The UDP header has a length of only 8 bytes. The calculation of a checksum is optional. A UDP checksum value of 0 means for the recipient that no checksum has been calculated in the sender.

The following figure shows a comparison between a UDP and a TCP data package, both embedded in an IP datagram and then packed in the Ethernet frame. Further reductions of the maximum possible user data length result from the headers of the higher protocol and application layers.

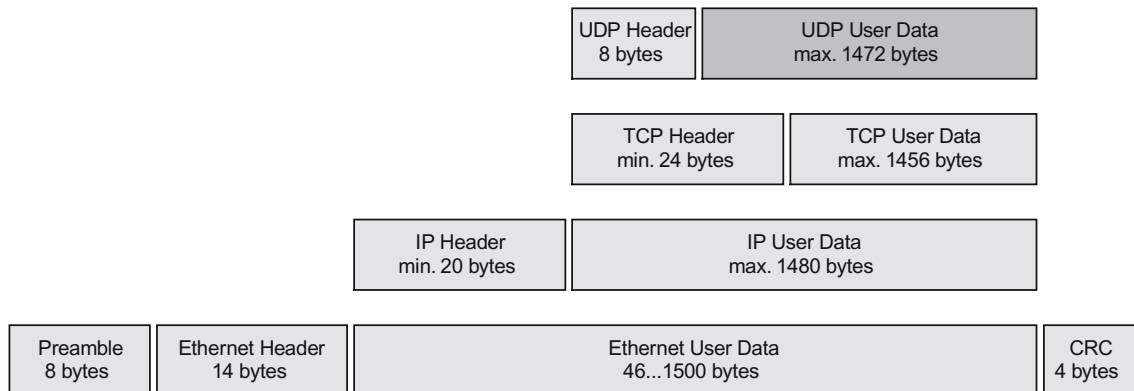


Figure 1-12: Comparison between UDP and TCP data packages

1.3.6 OpenModbus on TCP/IP

OpenModbus is a protocol of the application layer above TCP/IP. The basis for (Open-)Modbus is provided by the Modbus RTU specification. Modbus RTU is a standardized master-slave protocol for serial transmission lines which allows the master to read and write bit (coil) and word values (register) from/to the slaves (see also reference to the existing documentation "Modbus Telegrams").

The major field of application for the Modbus protocol is the fast data exchange between the automation devices as well as between visualization systems (HMI / SCADA) and automation devices.

OpenModbus on TCP/IP uses port 502 by default.

Telegram structure:

Modbus telegrams are composed of a transmission path-independent telegram part (simple Protocol Data Unit, PDU) and a network- or bus-specific header. The PDU and the header together are the application data unit (ADU). The following figure shows the general structure of a Modbus telegram as well as the structure of an OpenModbus on TCP/IP telegram.

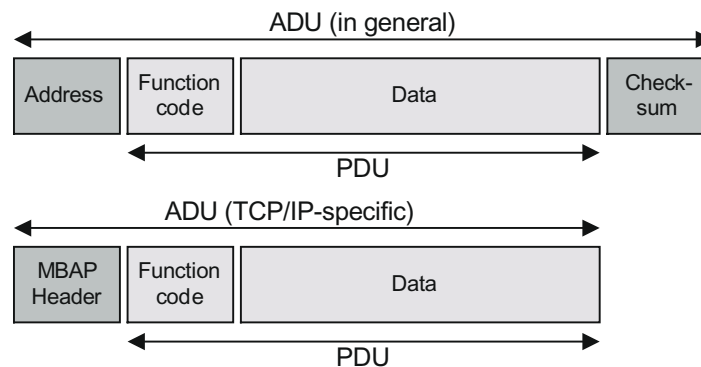


Figure 1-13: Structure of an OpenModbus on TCP/IP telegram

The telegram structure differs in some points from the original Modbus RTU format for serial transmission:

- The slave address at the beginning of the telegram is replaced by the 'Unit Identifier' field in the MBAP header (see below).
- The telegrams are structured in a way that the recipient is able to clearly identify their size. For some function codes the telegram length is fixed, so that it is possible to immediately determine the length from the function code. For function codes with variable telegram lengths the data part of the telegram contains an additional length specification.
- Since the IP header as well as the TCP header and the Ethernet frame provide their own checksum, the additional Modbus checksum is not required.

Modbus Application Protocol Header (MBAP)

The Modbus Application Protocol Header has a length of seven bytes and is structured as follows.

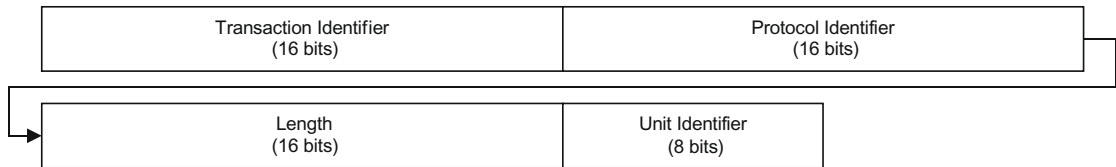


Figure 1-14: Structure of the modbus application protocol header

The 'Transaction Identifier' identifies a single process. The value is individually assigned by the master (client) on the occurrence of a request and copied into the respective response by the slave (server). This way the master is able to clearly assign an incoming telegram to the previous request even if there are several services running simultaneously.

The 'Protocol Identifier' is used to distinguish different future protocols. A Modbus master always sets this value to 0. A slave copies the value in the request into the header of the response.

The length field specifies the total length of the subsequent fields including the 'Unit Identifier' and the user data in bytes. The length is calculated by a master as well as by the slave when a telegram is generated.

The 'Unit Identifier' is used for routing purposes. Typically it is used for the conversion of a telegram by a gateway, if the master is located in a TCP/IP network and connected via this gateway to a serially connected Modbus or Modbus+ slave.

1.3.7 BootP and DHCP

The addresses on the lowest layers of a network arise from the used network technology. In case of Ethernet these are the MAC addresses which are predetermined in each device. Above layer 2 the IP addresses are established. IP addresses are freely assigned depending on the network structure. By definition, each device within a TCP/IP network must have a unique IP address before it can send or receive data via the network. If a device is able to permanently store the IP address once it has been set, the protocol software is initialized with the stored IP address each time the device is booted later. If a device is not able to store its IP address or if it still has the delivery settings, the device is booted without a valid IP address. For such cases, protocols exist which allow the assignment of IP addresses via the network. The best-known representatives of these protocols, BootP and DHCP, are described in the following sections.

Bootstrap Protocol (BootP):

The bootstrap protocol is specified in RFC 951. As an application it is directly based on UDP. BootP works in accordance to the client-server principle. It uses port 67 for the communication with the server and port 68 for the communication with the client. The BootP client communicates with the server by exchanging one single data package. Aside from the IP address of the computer, this data package can additionally contain the IP addresses of the next router and a particular server as well as the name of the boot file. By means of an other manufacturer-specific input-output map additional information can be entered, for example the subnet address and the domain name server.

In the initial state the client first has no valid IP address. To be able to send a request to the BootP server via IP, the client first uses the special IP address (broadcast). Among other things, this request contains the MAC address of the client. The response from the BootP server is also sent as IP broadcast. By comparing the received MAC address with its own address, a client determines whether this data package is directed to it. This way the client is informed about his own IP address and other things. The assignment of an IP address (and the additional information belonging to that) to a MAC address is performed in the BootP server using static tables administered by the network administrator.

Since the Bootstrap protocol is based on UDP, the transmission safety mechanisms known from TCP are not available. BootP has to be able to detect possible errors on its own. Each time it sends a BootP request, the client starts a retransmission timer. If it doesn't receive a BootP reply before the timer has elapsed, it generates the data packages again. To avoid that all clients simultaneously send their requests to the BootP server (e.g. after a voltage breakdown) and thus cause a server overload, the BootP specification recommends to use a random value for this timer.

Aside from the possibility to poll its own IP address, the bootstrap protocol can also be used by clients that already know their IP address to poll additional information, e.g. Name Server.

Dynamic Host Configuration Protocol (DHCP):

Today, many bigger companies have complex and sometimes global TCP/IP networks. The set-up and maintenance of such large networks is very expensive. Moves of individual employees or entire departments normally require to change the IP addresses and other parameters, e.g. the subnet mask. Furthermore already in the beginning nineties worries came up that a bottleneck could arise in the assignment of free IP addresses. For these reasons the Dynamic Host Configuration Protocol (DHCP) was developed on the basis of the existing bootstrap protocol and published in 1993 in the RFC 1541. DHCP enables a central administration and maintenance of all TCP/IP configuration parameters by a network administrator and thus allows to build up a plug-and-play TCP/IP network. In contrast to BootP which only allows the static assignment of network addresses, DHCP supports three alternative ways of IP address assignment.

Automatic IP address assignment

With the automatic assignment any fixed IP address is assigned to a client. On the initial login of the client to the server, the client is provided with a free address. This assignment is stored in the server. Then this IP address is assigned to the client each time it logs in to the server. This IP address cannot be used by another computer even if is not needed by the client, e.g. because it is temporarily out of the network. Compared with the method of setting the IP address directly on the device itself, this method has the disadvantage that for example a connection between two devices cannot be implemented until both devices have logged in to the DHCP server at least one time since the IP address is not known before.

Dynamic IP address assignment

The dynamic address assignment assigns an IP address to the client only for a specific period of time when it logs in to the server. The client can also release this address on his own initiative before the time is over if it doesn't need it any more. The advantage of this method is that an IP address which is no longer needed by a client can be assigned to any other client. In practice this could for example be used for an office that has less IP addresses than employees and where always only a part of the employees is present. A disadvantage of this method is that no fixed connections can be implemented between network nodes using IP addresses since the IP address can change with each login of a client. In practice this could for example be relevant for an installation with several controllers that exchange the process data via corresponding function blocks in the user program.

Manual IP address assignment

Using the manual assignment the network administrator is able to explicitly assign an IP address to a client by defining corresponding static tables in the server. This means that the administrator already defines in advance which configuration he wants to assign to which client. In this case DHCP is only used as a means of transport. A disadvantage is that the IP addresses cannot be used flexibly. An advantage arises from the fact that the IP addresses of all devices are known from the beginning and can be taken into account for the implementation of fixed communication connections already during the planning phase.

Function

DHCP also supports the BootP relay agents defined in the bootstrap protocol. The BootP relay agents have the task to forward DHCP messages from and to network segments which do not have an own DHCP server. Therefore, relay agents to a certain extent carry out the functionality of a router on the DHCP level. The use of relay agents has the advantage that not every subnet requires its own DHCP server.

The processes of the DHCP protocol are implemented using seven different types of messages.

Type of message	Description
DHCP Discover	Client broadcast to localize the available DHCP servers in the network.
DHCP Offer	Broadcast or unicast response of a server to a discover message with configuration parameters for the client.
DHCP Request	Broadcast of a client to all servers to request the offered parameters from a dedicated server with the simultaneous rejection of the parameters of all other servers.
DHCP Ack	Message from a server to a client, includes configuration parameters and the IP address.
DHCP Nak	Message from a server to a client, rejection of the request for configuration parameters and IP address.
DHCP Decline	Message from a client to a server saying that the configuration parameters and the IP address are invalid.
DHCP Release	Message of a client to a server saying that the IP address is no longer needed and available for use again.

Table 1-5: Message types of the DHCP protocol

Below the initialization process for a dynamic assignment is explained from the client's point of view.

After the boot process, the client is first in the initial state and sends a DHCP Discover message within its local subnet (subnet broadcast). This message can optionally already include a suggestion of the client for the required parameters. This can be the case if the client has already received corresponding values during a previous process and stored this. In any case the client has to hand over the hardware address (MAC address) since at this point in time it is not yet clear whether the client already has a valid IP address and can be accessed via this address. Since not every subnet necessarily has its own DHCP server, the DHCP Discover message is also forwarded to further subnets via possibly existing relay agents.

After it has sent the DHCP Discover message, the client is waiting for configuration offers in the form of DHCP Offer messages from DHCP servers that are waiting for its request. The servers try to directly respond to the client's request with a unicast. However, this is only possible if the client already has an IP address assigned (e.g. from a previous session) the lease period (assignment period) of which is not yet expired. Otherwise a DHCP server sends its message to the broadcast address 255.255.255.255. Aside from the available network address (IP address) for the client and the further communication parameters this message also contains the hardware address (MAC address) previously transmitted by the client with the DHCP Discover message. This way the client is able to determine whether the DHCP Offer message of a server is directed to it.

While the client is now waiting for the DHCP Offer message from the servers, a timer is running. If the server doesn't receive a DHCP Offer until the timer has expired, the DHCP Discover message is repeated. If the client received one or more DHCP Offer messages from one or several servers, it in turn responds with a DHCP Request message as a broadcast. This

message contains a server identifier with the designation of the DHCP server the parameters of which the client has chosen. The message is sent as a broadcast to inform all servers about which server the client has decided for. The servers which were not chosen by the client determine this fact by comparing the 'server identifier' field with their own designation and then interpret the DHCP request as a rejection. As a result, they release the parameters temporarily reserved for the client which can then be used for requests by other clients.

The server chosen by the client with the DHCP Request then responds with a DHCP Ack message containing all configuration parameters for the client including the lease period. The client then performs parameter testing (e.g. ARP for the assigned IP address, refer to 0). If the test is successful, the configuration of the client is finished. The client is now able to send and receive TCP/IP packages. If, however, the client detects faulty parameters in the DHCP Ack message, it sends a DHCP Decline message to the server and then repeats the configuration process. The configuration process is also started again if the client receives a DHCP Nak message from a server instead of a DHCP Ack message.

After a successful configuration the client permanently verifies the expiry of the lease period and, if necessary, sends a DHCP Request message to the configuring server to renew its lease period. If, however, a client wants to finish its address assignment (prematurely), it sends a DHCP Release message to exactly this server.

1.3.8 Address Resolution Protocol (ARP)

On the physical layer diverse mechanisms are used to address the individual devices in the network. For example, Ethernet uses a 48 bit long MAC address. During the installation of TCP/IP protocols a 32 bit long IP address is assigned to this hardware address. Since the physical layer is working completely independent of the above layers, the IP protocol does not know the hardware addresses of the communication parameters. This is why the Address Resolution Protocol (ARP) has been developed as an auxiliary protocol for layer 3. ARP converts IP addresses to the respective network-specific hardware addresses. This address mapping is performed using either the static or the dynamic method.

Static address mapping

For static address mapping the system administrator has to create tables in the used devices containing the fixed mapping between the hardware and the IP address. To setup a connection for a device configured this way, first the respective entry with the IP address of the target node is searched in the table. If a corresponding entry is found, the connection can be established. The use of the static method makes only sense in small networks since each modification of the network or a device requires the tables of all devices to be updated.

Dynamic address mapping

Today, the most common method of converting IP addresses to hardware addresses is the dynamic method. Before data are transmitted via the network, the Internet protocol verifies whether the entry for the target node is present in the ARP address table (ARP cache) of the device. If no corresponding entry can be found, the wanted address is requested from all existing computers in the network using a broadcast (ARP Request). Only those computers respond which have an entry for the wanted IP address. This is at least the wanted computer itself. The response (ARP Reply) is stored in the local ARP cache of the requesting device. The table entry first only exists for 20 minutes before it is deleted by the ARP timer. Each further use of the entry starts the timer again. The local storage of the combination of IP and hardware addresses considerably reduces the access to the network. The time limitation prevents the table size from growing caused by entries which are no longer needed. This way it is furthermore guaranteed that the respective entries are checked and/or updated from time to time. If, for example a computer is replaced by another one with the same IP number but a different hardware address, the ARP entries about the replaced computer in the other devices are invalid until the timer expires.

1.3.9 Other protocols and applications

Aside from the ones described above, numerous further protocols and applications exist in TCP/IP networks. However, since these are not part of the basic functionality and only optionally supported by some devices, they are not described in detail here. In the following sections only some of the best-known representatives of the higher protocols and applications are briefly described.

Hypertext Transfer Protocol (HTTP):

The Hypertext Transfer Protocol (HTTP) is used to exchange hypertext documents between www clients and www servers. Thus, HTTP which is already used as a standard protocol since 1990 is a communication protocol of the application layer. HTTP is based on a connection-oriented transport mechanism. If a client wants to download the data identified by a URL (e.g. www.abb.com/index.html) from a server, it establishes the TCP connection to the server via the logic port 80. The procedure for a communication between a www client (e.g. a PC) and a www server is always as follows:

1. The client establishes a TCP/IP connection to the respective server.
2. After the connection has been established completely, the client transfers the HTTP requests to the server.
3. The server responds to the requests and transfers the corresponding sets of data to the client.
4. Finally the TCP/IP connection is completely closed again.

Thus, HTTP is a request/response protocol. The www client transfers the HTTP request to the server. The server reacts with a HTTP response consisting of the response header (status line, used HTTP version, status/error message, server information) and the actual message (message body).

File Transfer Protocol (FTP:)

The File Transfer Protocol provides a standardized interface for the exchange of files between computers, independent of their design and operating system as well as for the administration of stored data. FTP is directly based on TCP and described in RFC 959 and the MIL-STD 1780 specifications. The FTP protocol uses port 21 for the control connection and port 20 for the data connection.

Communication between a FTP client (client PI, PI = Protocol Interpreter) and a FTP server (server PI) is performed using a set of specific commands with corresponding acknowledgements. Generally a command is composed of a four-figure character string (e.g. STOR) followed by additional information (e.g. specification of the path). Acknowledgements consist of three-figure character strings with optional text.

The following figure shows the FTP command syntax.

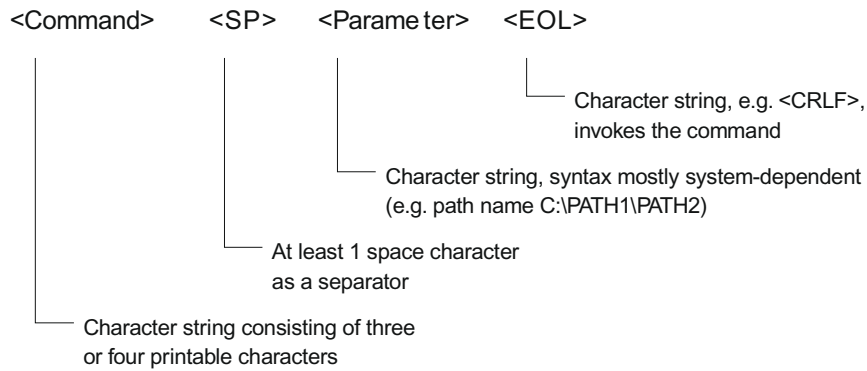


Figure 1-15: FTP command syntax

Common FTP implementations have a uniform and partly graphical user interface. So, for example many applications have the commands GET for transferring a file from a remote computer to the local computer and PUT for transferring a file from the local computer to the remote computer. These two commands are not defined by FTP. However there are existing FTP commands with corresponding functions (GET -> RETR, PUT -> STOR). Consequently it's the application's task to convert user commands to corresponding FTP commands. For the user itself, this conversion is not visible.

Simple Mail Transfer Protocol (SMTP):

The Simple Mail Transfer Protocol enables the transmission of Email messages. SMTP is directly based on TCP and uses the well-known port 25. The protocol is described in RFC 821 and the MIL-STD 1781 standard. SMTP only enables the transmission of Emails via a data network. It is not determined how the message is switched between the user and SMTP nor how the received message is stored and presented to the user. These tasks have to be done by the higher applications.

Messages sent by means of SMTP normally have the format defined in RFC 822. A 822 message is composed of some lines of header information followed by an empty line as a separator and the text. This text is also called the message body. Normally a header line consists of a keyword, a colon and a value (text, address) for the respective keyword. Examples of typical keywords are "from", "to" or "subject". They describe the sender and the recipient of the message as well as its subject. SMTP allows to specify several recipients one behind the other to simultaneously send a message to several recipients. The message body consists of pure text (7 bit ASCII) of any length. The following example shows an SMTP message in 822 format.

```
FROM: Paul Anybody <Paul@Anybody.COM>
SUBJECT: Confirmation of appointment
TO: Jack.Sample@Test.DE
```

Date on Thursday, 12 a.m. is ok.

Paul

Network Information Service (NIS):

The Network Information Service serves for the centralized administration and maintenance of a local network and its users. This is implemented by several databases which are used to administer the computer names, user groups, services and user accounts. NIS furthermore allows to structure a local network and its users into smaller administration units, the so-called domains. Not least because of the missing data encryption and various safety deficiencies, the NIS protocol has later been further developed to NIS+.

Domain Name Service Protocol (DNS):

Prior to the introduction of the Domain Name Service a central host table (hosts.txt) has been administered by the Network Information Center of the Defense Data Networks. The goal of development of the protocol was to replace this table by a distributed database application in order to ease the maintenance of the host and domain names and to reduce and distribute the load caused by the transmission of host tables in the tremendously growing Internet.

DNS is based on UDP (port 53). The domain names used in the Domain Name Service are based on a hierarchical naming concept. A domain name consists of several subnames separated by dots (e.g. sample.example.com). The Domain Name System is structured like a tree, starting from the so-called root. Each node in this tree is a zone (domain). A Domain Name Server always only knows the next higher and the next lower server of the respective domain. A Domain Name Server is responsible for one complete zone. A zone begins in a node and contains all branches included under this node. The highest domain level is called the Top Level Domain. The following figure shows the Top Level Domains for the USA.

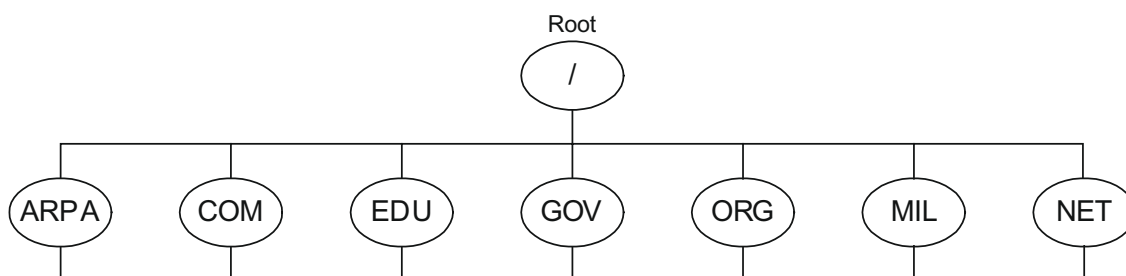


Figure 1-16: Top Level Domains of the USA

The task of a Domain Name Server is to deliver a requesting client computer the IP addresses for a symbolic name.

Internet Name Service (IEN 116):

The Internet Name Service protocol is the oldest name service protocol. It was published as early as 1979 as Internet Engineering Note 116 (EIN 116). This protocol is directly based on UDP and converts logical symbolic names into IP addresses.

If a computer wants to establish a connection to another device in the network but the user only knows the logical device name (e.g. controller1) but not its IP address, the computer first transmits a name request containing the logical device name to the name server. The name server then delivers the IP address of the device and by means of this IP address the client is now able to establish the connection to the device. The name server has to be known by the local computer. This is normally set by the responsible system administrator in a name server file.

Hosts file:

In the previous sections different naming services are described which allow to determine the IP address of a device by means of its symbolic name. However, if no name server is available in a network or if the device to be called is not registered in the name server, using the host table of the local computer still represents a simple possibility to call the respective device using its symbolic name.

On Windows computers the hosts file is normally located in the subdirectory ...etc (e.g. for Windows NT C:\WINNT\SYSTEM32\DRIVERS\ETC\HOSTS). This file is a simple text file containing the assignment of IP addresses to the host names and can be edited using any editor. By default the hosts file at least contains the loopback address (127.0.0.1 localhost) and, if applicable, the IP address of the local computer. This list can be expanded as desired. For this purpose the entries must have the following structure: IP address of the computer, followed by the respective host name and optional alias names. The IP address and the host name (as well as the alias names) have to be separated by at least one blank character. The # character identifies the beginning of a comment. The end of a comment is automatically defined by the line end. Comments are ignored by the system when interpreting the hosts file.

The following example shows a hosts file.

```
#Loopback address for localhost
127.0.0.1 localhost

#System part 1
193.0.4.1 Basic unit
193.0.4.2 Visualization      #Control room
193.0.4.3 Controller1       Silo
193.0.4.4 Controller2       Pumping station

#System part 2
193.0.5.1 Mixer
193.0.5.2 Heating
193.0.5.3 Control cabinet
```

1.4 Cabling

1.4.1 Network cables

There are numerous standards regarding the Ethernet cabling which are to be observed when setting up a network. So, the standards TIA/EIA-586-A and ISO/IEC 11801 define the properties of the cables to be used and divide the cables into categories. While the TIA/EIA specification is more directed to the American market, the ISO/IEC standard meets the international requirements. Furthermore the European standard EN 50173 which is derived from ISO/IEC 11801 represents the standard to be observed for structured cabling.

Usually twisted-pair cables (TP cables) are used as transmission medium for 10 Mbit/s Ethernet (10Base-T) as well as for 100 Mbit/s (Fast) Ethernet (100Base-TX). For a transmission rate of 10 Mbit/s cables of at least category 3 (IEA/TIA 568-A-5 Cat3) or class C (according to European standards) are allowed. For Fast Ethernet with a transmission rate of 100 Mbit/s, cables of category 5 (Cat5) or class D or higher have to be used. The following table shows the specified properties of the respective cable types per 100 m.

Parameter	10Base-T [10 MHz]	100Base-TX [100 MHz]
Attenuation [dB / 100m]	10.7	23.2
NEXT [dB / 100m]	23	24
ACR [dB / 100m]	N/A	4
Return loss [dB / 100m]	18	10
Wave impedance [Ohms / 100m]	100	100
Category	3 or higher	5
Class	C or higher	D or higher

Table 1-6: Specified cable properties

The TP cable has eight wires where always two wires are twisted to one pair of wires. Different color codes exist for the coding of the wires where the coding according to EIA/TIA 568, version 1, is the most commonly used. In this code the individual pairs are coded with blue, orange, green and brown color. While one wire of a pair is single-colored, the corresponding second wire is colored alternating with white and the respective color. For shielded cables it is distinguished between cables that have one common shielding around all pairs of wires and cables that have an additional shielding for each pair of wires. The following table shows the different color coding systems for TP cables:

Pairs	EIA/TIA 568 Version 1		EIA/TIA 568 Version 2		DIN 47100		IEC 189.2	
	1	2	3	4	5	6	7	8
Pair 1	white/blue	blue	green	red	white	brown	white	blue
Pair 2	white/orange	orange	black	yellow	green	yellow	white	orange
Pair 3	white/green	green	blue	orange	gray	pink	white	green
Pair 4	white/brown	brown	brown	slate	blue	red	white	brown

Table 1-7: Color coding of TP cables

Two general variants are distinguished for the pin assignment of the normally used RJ45 connectors: EIA/TIA 568 version A and version B where the wiring according to EIA/TIA 568 version B is the most commonly used variant.

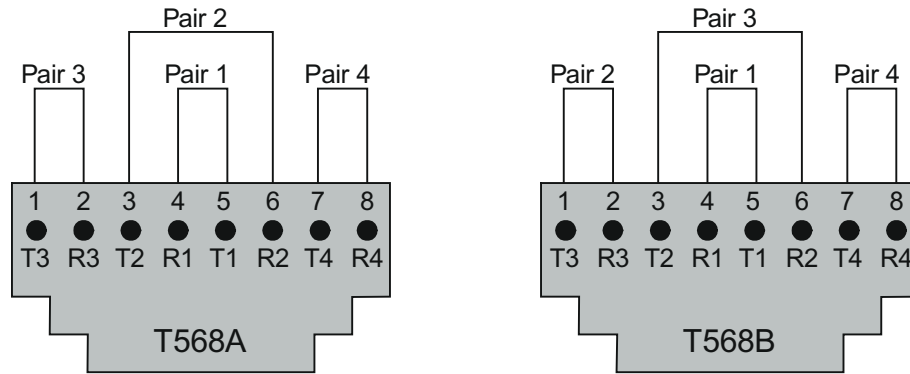


Figure 1-17: Pin assignment of RJ45 sockets

1.4.2 Connector pin assignment

Today, RJ45 connectors are accepted as a standard for the cabling of 10 Mbit/s networks (10Base-T) as well as for 100 Mbit/s networks (100Base-TX). Generally, the same pin assignment is used for both variants.

Pin number	Signal
1	TD+ (data output)
2	TD- (data output)
3	RD+ (data input)
4	-
5	-
6	RD- (data input)
7	-
8	-

Table 1-8: Pin assignment of RJ45 connectors

1.4.3 1:1 cables and crossover cables

A direct connection of two terminal devices is the simplest variant of an Ethernet network. In this case a so-called crossover cable (also called crossconnect or crosslink cable) has to be used to connect the transmission lines of the first station to the reception lines of the second station. The following figure shows the wiring of a crossover cable.

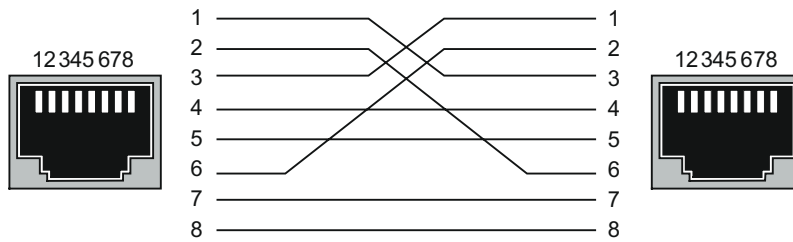


Figure 1-18: Wiring of a crossover cable

For networks with more than two subscribers, hubs or switches have to be used additionally for distribution (see also section 1.4 "Cabling"). These active devices already have the crossover functionality implemented which allows a direct connection of the terminal devices using 1:1 cables.

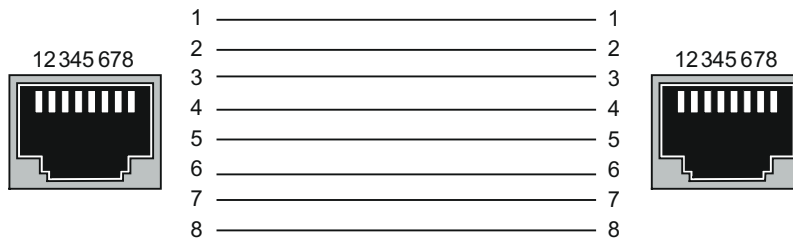


Figure 1-19: Wiring of a 1:1 cable

1.4.4 Cable length restrictions

For the maximum possible cable lengths within an Ethernet network various factors have to be taken into account. So, for twisted pair cables (for transmission rates of 10 Mbit/s and 100 Mbit/s) the maximum length of a segment which is the maximum distance between two network components is restricted to 100 m due to the electric properties of the cable.

Furthermore the length restriction for one collision domain has to be observed. A collision domain is the area within a network which can be affected by a possibly occurring collision (i.e. the area the collision can propagate over). This, however, only applies if the components operate in half duplex mode since the CSMA/CD access method is only used in this mode. If the components operate in full duplex mode, no collisions can occur. Reliable operation of the collision detection method is important which means that it has to be able to detect possible collisions even for the smallest possible frame size of 64 bytes (512 bits). But this is only guaranteed if the first bit of the frame arrives at the most distant subscriber within the collision domain before the last bit has left the transmitting station. Furthermore the collision must be able to propagate to both directions within the same time. Therefore, the maximum distance between two ends must not be longer than the distance corresponding to the half signal propagation time of 512 bits. Thus, the resulting maximum possible length of the collision domain is 2000 m for a transmission rate of 10 Mbit/s and 200 m for 100 Mbit/s. In addition, the bit delay times caused by the passed network components have also to be considered.

1.5 Network components

The topology of an Ethernet network is like a star or tree structure. Up to two stations can be connected to each segment where active distribution devices like hubs or switches are also considered as a station. The following figure shows an example of a simple Ethernet network.

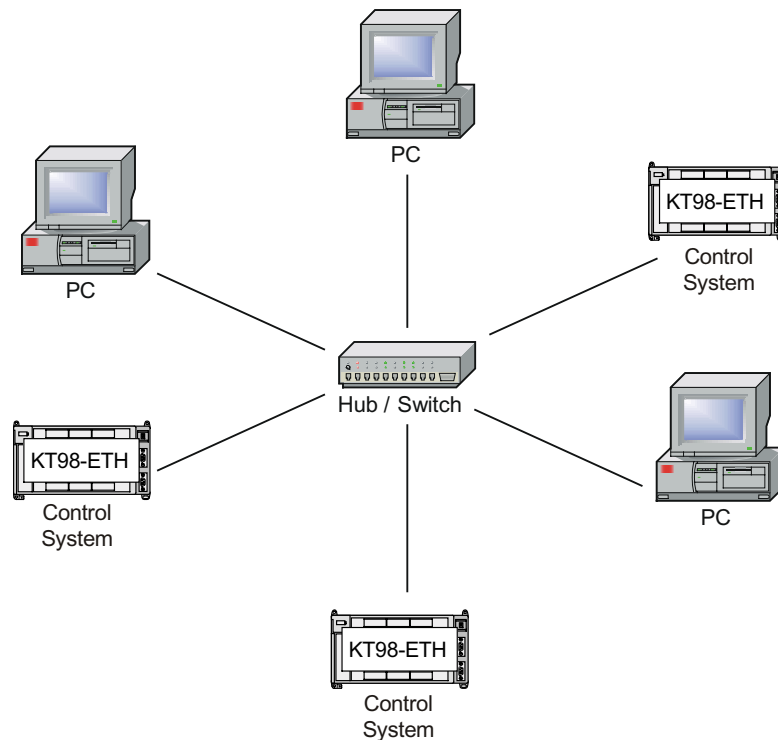


Figure 1-20: Example of a simple network

The following sections introduce the different types of components required for a network.

1.5.1 Terminal devices

Terminal devices are devices that are able to send and receive data via Ethernet, e.g. controllers with an Ethernet coupler or PCs with an integrated network adapter. With this, one of the essential functions of a network adapter is to transfer all data packages to the PC itself instantly and without any loss. Occurring defiles or even errors can cause data packages to be lost. Such losses of data have to be got under control by higher protocols (e.g. TCP/IP) which results in considerable performance reductions. The direct implementation of higher protocols on the network adapter can increase the performance and save the resources of the host system (e.g. controller).

1.5.2 Repeaters and hubs

At the dawning of the Ethernet, the repeaters had only two network connections and were used to connect two segments to each other in order to extend the segment length. Later, repeaters with more than two network connections were available. Those star distributors which are able to connect several segments are called hubs. Apart from the number of network connections the functionality of hubs and repeaters is identical. This is why we only use the term "hub" in the following descriptions.

Hubs are operating on the lowest layer of the ISO/OSI model and are therefore independent of the protocols used on Ethernet. The network connections of hubs are exclusively operated in half duplex mode. Due to this, collision domains can freely propagate beyond the hubs. A hub can only support one transmission rate for all connections. Therefore it is not possible to connect segments with different transmission rates via a simple hub. For this purpose a dual-speed hub has to be used. The fundamental functions of hubs are as follows:

- Restoration of the signal magnitude
- Regeneration of the signal timing
- Propagation of a detected collision
- Expansion of short fragments
- Creation of a new preamble
- Isolation of a faulty segment

When transmitted over the medium (e.g. a twisted pair cable) the data signal is attenuated. The task of a hub is to amplify an incoming signal in order to make the full signal magnitude available at the outputs again. Furthermore a distortion of the binary signal's on-off ratio (jitter) can occur during data transmission. When transmitted via a hub, the hub is able to restore the correct on-off ratio of the signal which avoids the propagation of signal jitter beyond the segment.

However, one of the most important tasks of a hub is to propagate occurring collisions within the entire collision domain so that the collision can be detected by all connected stations. If it detects a collision on one of his connections, the hub sends a so-called jam signal over all connections. If a hub receives a data fragment which could, by its principle, only be created by a collision, it first brings the fragment to a length of 96 bits and then forwards it via the ports. This shall guarantee that the data fragment can be received by all stations independent of their distance to the hub and removed from the network. The detected data fragments are removed by the terminal devices by not forwarding them to the higher layers.

By means of the data package preamble the beginning of a data package is detected so that the recipient can synchronize to the incoming data stream. However, during the data transmission it can occur that the first bits of a preamble are lost. The task of the hub is to restore a possibly incomplete preamble before forwarding it.

If collisions occur within one segment in large numbers in a short period of time or if e.g. a short circuit on a data line causes failures, the hub switches off the faulty segment to avoid interference to the entire collision domain.

10 Mbit/s hubs:

The 10Base-T connections of a 10 Mbit/s hub are implemented as MDI-X ports and therefore already crossed internally. The advantage is that the terminal devices can be directly connected using 1:1 twisted pair cables and no crossover cables are required. Some hubs additionally have a so-called uplink port which can be used to connect another hub. In order to also enable the use of a 1:1 cable for this port, it is implemented as a normal non-crossed MDI port. In many cases this port can also be switched between MDI and MDI-X or is implemented as a double port with two connections in parallel (1 x MDI, 1 x MDI-X). In this case it has to be observed that these parallel ports may only be used alternatively and not at the same time.

Hubs are normally equipped with several LEDs for status indication. So, for example a Link LED indicates the correct connection between the terminal device and port at the hub. This way, incorrect cabling can be quickly detected. Further LEDs indicate for example the data traffic over a port or the collisions.

The maximum permitted number of 10 Mbit/s hubs within one collision domain is limited to 4. This restriction is due to two reasons. One reason is that the bit period time delay which is inevitably increased by each hub must not exceed 576 bit periods. The second reason is that the so-called interframe gap (IFG) must not be shorter than at least 47 bit periods. The interframe gap describes the time interval between two data packages and shall allow the receiving stations to recover from the incoming data stream. However, the regeneration of an incomplete preamble performed by the hub reduces the time between the data packages due to the completion of possibly missing bits.

One possibility to get round the restriction to four hubs is the use of stackable hubs. These hubs are connected to each other via a special interface instead of using the uplink port and therefore constitute one logic unit. As a result, they appear as one single big hub to the external.

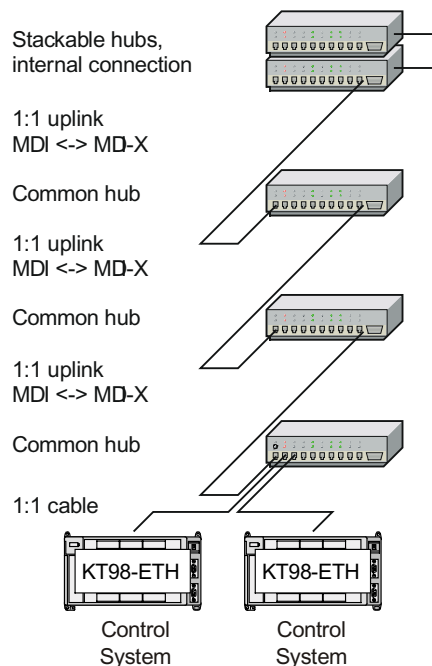


Figure 1-21: Stackable hubs

100 Mbit/s hubs:

The principle operation of 100 Mbit/s hubs is like the 10 Mbit/s hubs. However, the hubs for 100 Mbit/s Ethernet have to be additionally distinguished to class I and class II hubs.

Class I hubs (or class I repeaters) are able to connect two segments with different transmission media. For this purpose the complete data stream has to be decoded on the receiving side and encoded again on the transmission side according to the transmission medium. This conversion process leads to higher delay times. Due to this, only one class I hub is permitted within one collision domain.

In contrast, class II hubs support only one transmission medium. No conversion of the data stream is required. This leads to shorter delay times compared with class I hubs. This is why for two segments with a maximum length of 100 m each, up to two class II hubs which are again connected to each other via a 5 m long segment can be used within one collision domain.

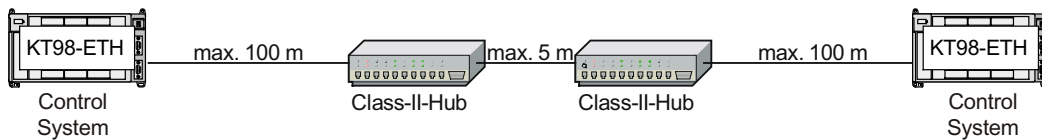


Figure 1-22: Use of a class II hub

10/100 Mbit/s dual-speed hubs

In contrast to the simple hubs, dual-speed hubs are able to support two transmission rates and thus enable to connect two Ethernet networks with different data rates to each other. Dual-speed hubs are internally structured like two separate hubs or paths (one for each data rate). By means of the auto negotiation function the transmission rate of the connected station is determined and automatically switched to the corresponding path. Each internal path is a separate hub. For the temporary storage of the data packages the paths are connected to each other via an internal switch. Dual-speed hubs likewise operate in half duplex mode. However, the internal switch provides a clear separation of the 10 Mbit/s and the 100 Mbit/s side so that unlike the simple hubs a collision domain cannot reach beyond the borders of the corresponding side of the dual speed hub.

1.5.3 Bridges, switches and switching hubs

Basically the terms bridge, switch and switching hub designate the same. In the early beginning of the Ethernet the term bridge was formed by the fact that a bridge had only two network connections. Later, so-called multiport bridges with several connections came up which were also called switches or switching hubs. This is why we use the common term "switch" in the following descriptions for all the components mentioned above.

The use of a switch is another variant of connecting network segments to each other. The decisive difference between a hub and a switch is that a switch is operating on the second layer of the ISO/OSI model, the MAC layer.

The following sections describe the functionality of such a layer 2 switch. For reasons of completeness it has to be mentioned that switches operating on higher and therefore protocol-specific layers also exist.

Using a switch, load separation between networks can be implemented which leads to an increased performance due the reduced load of the individual segments. In contrast to a hub, the switch is not operating transparently (i.e. it doesn't forward all data packages via all ports) but decides on the basis of the MAC target address whether and via which port an incoming data package has to be forwarded. The data package is only forwarded if the target station is located in another segment or if the target address of the data package contains a multicast or broadcast address.

As already mentioned, the decisive advantage of a switch is the logical separation of networks. Therefore, a switch represents a border for a collision domain. Aside from the performance improvement, the use of a switch allows a network to be extended beyond the usual borders.

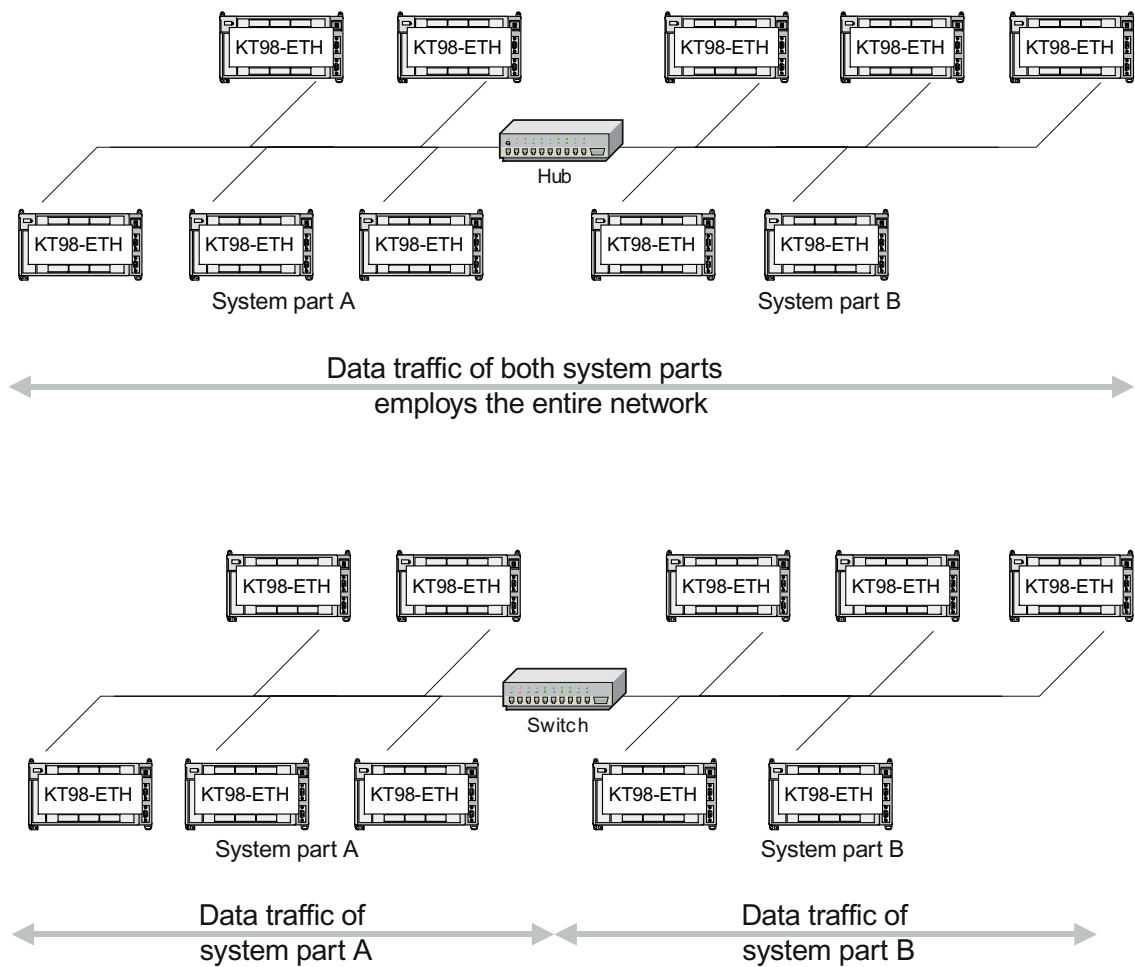


Figure 1-23: Use of hubs and switches

To enable crosswise traffic between the segments, a switch has to be able to temporarily store the incoming data packages until they can be transmitted on the forwarding segment. The decision about forwarding of data packages is done using address tables. These address tables are generated by the switch itself during a self-learning process. During this process, the switch remembers the source addresses (MAC addresses) of incoming data packages of a port. If it later receives further data packages, the switch compares their target addresses with the entries in the address tables of the ports and, in case of a match, forwards the respective package via the corresponding port. Here, the following cases have to be distinguished:

- If the source and the target station are located within the same segment, the data package is not forwarded.
- If the station of the target address is located in another segment than the source station, the data package is forwarded to the target segment.
- Data packages containing a multicast or a broadcast address as the target address are forwarded via all ports.
- A data package with a target address which is not contained in the address tables is forwarded via all ports (Frame Flooding).

The latter case normally only occurs during the first time after starting a switch since the address is usually entered after some time through the exchange of a data package.

In order to limit the sizes of the address tables, addresses which are not used over a longer period of time are additionally removed from the tables. This also avoids incorrect forwarding as it would appear e.g. when a station is moved within the network.

To enable the building of a redundant network structure (as it is often found in more complex networks) using switches, the so-called spanning tree method has been introduced. With this method the switches exchange configuration messages among themselves. This way the optimum route for forwarding data packages is determined and the creation of endless loops is avoided. The exchange of messages is performed cyclic. As a result a connection breakdown is detected and forwarding is automatically changed to another route.

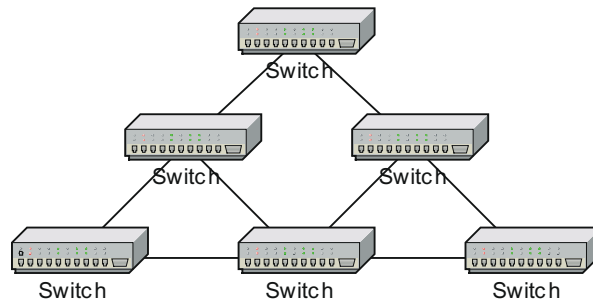


Figure 1-24: Redundant network structure using switches

Using a switch instead of a hub increases the bandwidth of the individual segments and therefore leads to an increased performance. Building a network consistently with switches furthermore enables full duplex operation and thus simultaneous data traffic in both directions since switches are able to establish dedicated peer-to-peer connections between the individual ports. The use of the access method CSMA/CD is not required since collisions can no longer occur. Depending on the network structure, this can further increase the performance drastically. For full duplex connections furthermore no length restrictions of the collision domain have to be observed.

1.5.4 Media converters

Media converters provide the possibility of connecting components to each other via different media. The most frequently occurring case for this is the conversion between twisted pair (TP) and fibre optic cabling.

When using media converters it has to be observed that a connected port operating in half duplex mode is no collision domain border. This is often not considered when using optical fibres to bridge a larger distance. The fibre optical port of a media converter furthermore does not support the auto negotiation function. Due to this, if an Ethernet component is directly connected to the fibre optic side of a media converter, the transmission mode has to be set fixed according to the component connected on the twisted pair side. If a connection between two twisted pair components is established using two media converters, it is absolutely required that both twisted pair components are operating with the same transmission mode. If necessary, manual setting has to be performed.

1.5.5 Routers

Routers connect networks with identical protocols or addressing mechanisms. The main task of a router is to perform the routing for the transmission of data packages from the sender to the recipient. Routers are able to effectively reduce the data traffic between individual networks by using different algorithms. The dynamic routing leads to a load reduction for the entire network. If the router has several alternative routes to the target station available, it will always choose the optimum way depending on the current load on the network and the expected costs.

In contrast to the switches which forward the packets on the basis of layer 2 (e.g. Ethernet), the routers operate on layer 3 (e.g. IP). While the switches forward the packets on the basis of the MAC addresses, the routers evaluate the contained IP addresses. For this purpose, when receiving a data package a router first has to remove the outer telegram frame in order to be able to interpret the addresses of the inner protocol and then it has to re-assemble the data package again before forwarding it. This results in higher latency periods (time of stay) of the data within the router itself. The investigation of a data package necessary for routing makes clear that a router has to be able to process all network protocols to be routed over this router. Due to the increasing spread of heterogeneous networks, today often routers are used which are able to support several network protocols (e.g. IP, IPX, DECnet, AppleTalk) instead of special IP routers. Such routers which are able to process several network protocols are called multi-protocol routers. Some routers additionally have a bridge functionality (bridge routers, Brouters) which enables them to also forward the data packages of protocols a router cannot interpret or which do not support the routing function (e.g. NetBios).

1.5.6 Gateways

A gateway is a computer which is able to couple completely different networks. Gateways are operating on a layer above layer 3 of the ISO/OSI model. They are used to convert different protocols to each other. For the connected subnetworks a gateway is a directly addressable computer (node) with the following tasks:

- Address and format transformation
- Conversions
- Flow control
- Necessary adaptations of transmission rates for the transition to the other subnetworks.

Gateways can furthermore be used to implement safety functions on the application layer (firewalls). For example, gateways are used for the coupling of PCs located in local area networks (LAN) to public long distance communications (wide area networks, WAN).

2.1 Features

2.1.1 Supported protocols

IP - Internet Protocol (RFC 791):

- Freely configurable IP address and network mask
- Configurable IP address of the standard gateway
- IP datagram size: 1500 bytes max.
- Route Cache size: 32 entries
- Route Timeout: 900 seconds
- Number of IP multicast groups: 64 for reception, unlimited for transmission

TCP - Transmission Control Protocol (RFC 793, RFC 896):

- Amount of user data for TCP telegrams: 1460 bytes max.

UDP - User Datagram Protocol (RFC 768):

- Amount of user data for UDP telegrams: 1472 bytes max.

BOOTP - Bootstrap Protocol (RFC 951, RFC 1542, RFC 2132):

DHCP - Dynamic Host Configuration Protocol (RFC 2131, RFC 2132):

OpenModbus:

- Client and/or server mode (several times)
- Up to 8 simultaneous client or server connections
- Supported function codes: 1, 2, 3, 4, 5, 6, 7, 15, 16
- Maximum amount of data per telegram: 100 coils (words) or 255 registers (bits)
- Configurable connection monitoring functions

NetIdent:

- Devices can be identified and accessed via the network (even unconfigured devices)
- Unique identification and localization via rotary switch on the devices

ARP - Address Resolution Protocol (RFC 826):

- ARP Cache size: 64 entries
- ARP Timeout: 600 seconds

ICMP - Internet Control Message Protocol (RFC 792):

IGMPv2 - Internet Group Management Protocol, version 2 (RFC 2236):

Further protocols and applications are in preparation

2.1.2 Sockets

- Number of sockets: 16
- Socket options can be set individually

2.1.3 Restrictions

- IP fragmentation is not supported
- TCP Urgent Data is not supported
- TCP port 0 is not supported
- TCP port 502 is reserved for OpenModbus
- TCP port 1200 is reserved for gateway access
- UDP port 67 is reserved for BOOTP and DHCP
- UDP port 25383 is reserved for NetIdent protocol
- UDP port 32768 is reserved for UDP blocks

2.2 Technical data

2.2.1 Technical data of the coupler

Coupler type	Ethernet coupler in PC/104 format
Processor	EC1-160, 48 MHz
Ethernet controller	EC1-160 internal
Internal power supply with	+5 V, 500 mA
Dimensions	96 x 102.2 x 23 mm
CE sign	yes

2.2.2 Interfaces

Ethernet	10/100 Base-TX, RJ45 socket
Serial	RS232, 8-pole miniDIN socket
LED indication	Status indication via 4 LEDs
Station identification	Rotary switch, 00..FF _{hex} (0..255 _{dec})

2.2.3 Technical data of the Ethernet interfaces

Transmission mode	Half or full duplex operation, adjustable
Transmission rate	10 or 100 Mbit/s, adjustable
Auto negotiation	optionally adjustable
MAC address	optionally configurable
Ethernet frame types	Ethernet II (RFC 894), IEEE 802.3 receive only (RFC 1042)

2.3 Connection and transfer media

2.3.1 Attachment plug for Ethernet cable

8-pole RJ45 plug

See also chapter 1.4 "Cabling".

Assignment:

Pin No.	Signal	Meaning
1	TxD+	Transmit data (line) +
2	TxD-	Transmit data (line) -
3	RxD+	Receive data (line) +
4	NC	Not used
5	NC	Not used
6	RxD-	Receive data (line) -
7	NC	Not used
8	NC	Not used
Shield	Cable shield	

Table 2-1: Pin assignment of the attachment plug for the Ethernet cable

2.3.2 Ethernet cable

For structured Ethernet cabling only use cables according to TIA/EIA-586-A, ISO/IEC 11801 or EN 50173 (see also chapter 1.4 "Cabling").

2.4 Ethernet implementation

2.4.1 Configuration

The Ethernet coupler is configured via PC using the 907 FB 1131 configuration software (refer to 907 FB 1131 documentation and chapter 4 "Design examples"). Configuration data created using 907 FB 1131 are not assigned directly to a project. Thus, the resulting file has to be downloaded separately to the controller (additionally to a 907 AC 1131 project) where it is stored in the Flash memory.

If the user defined project is written to SMC, the configuration data are automatically saved, too.

2.4.2 Running operation

The integrated protocols are automatically processed by the coupler and the operating system of the controller. The coupler is only completely ready for operation if it has been configured correctly before. Online access via the Ethernet coupler is available at any time, independent of the controller's state. Designable protocols (e.g. fast data exchange via UDP) require function blocks. They are only active while the user program is running. If the user program is stopped, all the planned connections possibly still existing are closed automatically.

2.4.3 Error diagnosis

The Ethernet coupler's operating condition as well as possible communication errors are always indicated by the LEDs next to the Ethernet connector. Malfunctions of the integrated protocol drivers or the coupler itself are indicated via the FKx error flags and the corresponding LEDs (refer to the error tables of the internal couplers). In case of designable protocols, information about occurring errors are additionally available at the outputs of the corresponding function blocks.

2.5 Diagnosis

Status LEDs:

Status indication via 4 LEDs:

LED	Color	Meaning
READY	yellow	The coupler is ready for operation.
RUN	green	Status of configuration and communication.
STA	yellow	Status of Ethernet communication.
ERR	red	Communication error

Table 2-2: Status LEDs

Ethernet error messages:

The Ethernet error messages are listed in section "Error messages of the internal couplers".

Function blocks:

In case of designable protocols (e.g. fast data exchange via UDP), information about possibly occurring errors are additionally available at the outputs of the corresponding function blocks.

Online diagnosis:

The field bus configuration tool 907 FB 1131 provides extensive online diagnosis functions (refer to documentation for the field bus configuration tool 907 FB 1131).

3 Designing and planning a network

3.1 Introduction

To obtain optimum performance within a network, it is absolutely essential to plan the network beforehand. This applies to both the initial installation as well as its expansion. Rashly installed networks can not only cause poor network performance, they even can lead to a loss of data since restrictions given by the standard are possibly not kept. At first glance, designing a network causes additional costs, but it will later reduce maintenance expenditures during operation.

The following sections shall explain some principle methods for determining a suitable network structure and give some hints how to find out the network utilization and performance.

3.2 Concepts for structuring a network

Three fundamental aims are to be considered when designing the concept of a network: Performance, quality and safety. The performance of a network is primarily described by the data throughput (as high as possible) and the transmission delay (as short as possible). Quality means stability, fail-safety and availability of the network. The safety aspect considers the safety of the transmitted data, i.e. protection against access to confidential data by unauthorized persons. Whereas performance and quality are planning goals for all kinds of networks, safety has mainly to be considered for networks which can be accessed from the "outside world". For example, a "closed" network inside an installation containing only automation components does not require particular protection of the data against unauthorized access.

Therefore, a detailed requirements analysis has to be done prior to the actual conception of a network in order to meet the specific requirements of the particular network.

Apart from planning the passive structured cabling, making a network conception also includes the selection of suitable active components such as hubs, switches or routers. Planning a new network starts with a registration of all systems (e.g. controllers) to be installed which shall be connected by the network and the requirements to the intended data exchange between these systems. When expanding or optimizing an existing network, first the actual situation has to be determined and the performance of the existing components with regard to the new requirements has to be assessed.

Regarding the network technology the following three general models are distinguished:

- Hierarchy model
- Redundant model
- Safe model

The selection of the suitable model as a basis for planning a network depends on the specific requirements of the installation. Office networks are typically built up based on the hierarchy model since the individual clients do not very often exchange data with each other but only periodically contact the server. Installation-internal networks which do not have any connection to the company network often only consist of automation devices and do not have a server. The connected controllers transmit data in short intervals directly to each other. Furthermore, the operational safety of installation-internal networks has a higher importance since data transmission malfunctions can result in incorrect behavior of the installation or even in production stops. In such cases it is more suitable to choose the redundant model or a safe model.

In the end, all three models shown above are based on the use of switching hubs (switches). Whereas in the past simple hubs were increasingly used to set up a network wherever permitted by the requirements, today almost exclusively switches are used. Using switches, historic Ethernet rules such as the length restrictions of a collision domain no longer have to be observed. This considerably simplifies the network design. Even though the use of switches could make us believe that networks can be expanded to an infinite size, it has to be considered that each switch involved in a data transfer causes a delay. Therefore, the IEEE-802.1d bridging standard recommends to limit the number of switches to be passed between two terminal devices to a maximum of seven switches.

3.2.1 Hierarchy model

The hierarchy model intends the subdivision of the network into several levels and a graduation of the data rate between the individual levels. For this purpose, normally at least two grades are used e.g. by connecting the server with a data rate of 100 Mbit/s to the network and the clients with 10 Mbit/s. The advantage of this design is that the server has 10 times the bandwidth of the clients available which enables the server to provide sufficient bandwidth and response time for several clients. Despite the fact that 10 times the bandwidth does not mean that 10 clients can simultaneously access the server, the data transmitted to or from the clients do only need one tenth of the time. In total, this reduces the response time for each single client.

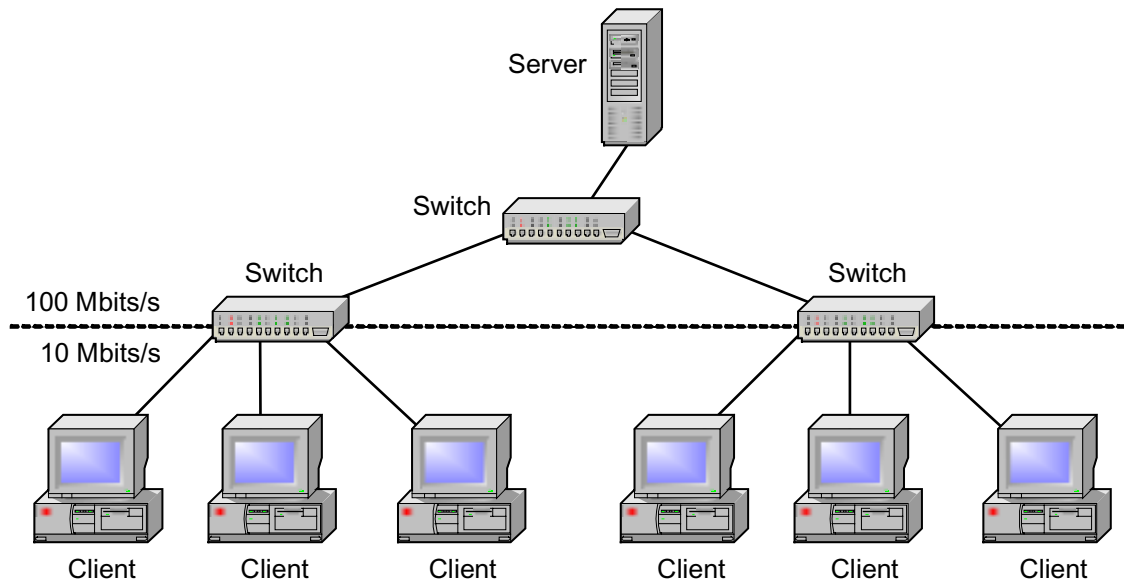


Figure 3-1 : Hierarchy model

When dimensioning the individual levels the utilization of the particular level has to be considered. Devices connected to each other via hubs can only be operated in half-duplex mode. Consequently they have to share the commonly used network (shared media). If the utilization of such a shared media is higher than 40 % over a longer period of time, a switch should be used instead of a hub in order to subdivide and thus relieve the collision domain. The utilization threshold within such a switched media is 80 %. If this value is exceeded the utilization should be reduced by selecting a smaller grouping.

3.2.2 Redundant model

The meshed Ethernet structure is a typical example for a redundant network model. To obtain fault tolerance, several connections have to be established between switches or nodes. This way, data exchange can be performed using another (redundant) connection if one connection fails.

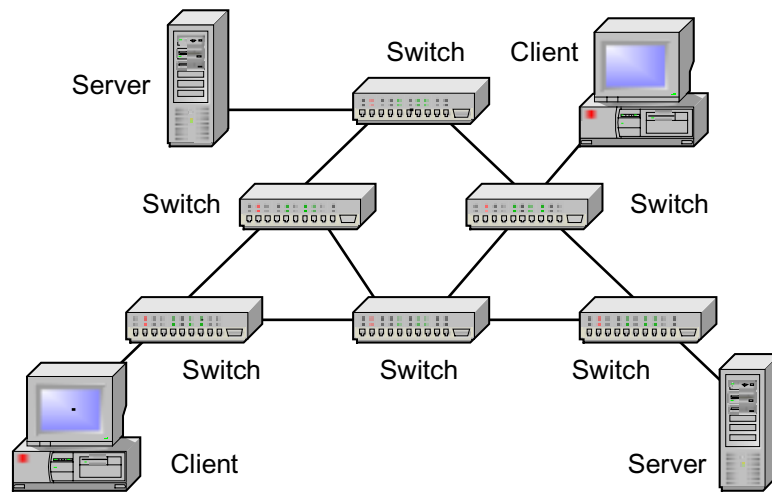


Figure 3-2 : Redundant model

However, this meshed constellation leads to loops which would make well-ordered data exchange impossible. The loops would cause the broadcast or multicast data packages to endless stray in the network. In order to suppress such loops the spanning tree mechanism (refer to 1.5.3 "Bridges, switches and switching hubs") is used which always activates only one unique connection and deactivates all other possible connections. On the occurrence of a fault (e.g. caused by an interruption of the network line) the redundant connection is re-activated and then maintains communication between the switches. However, switching of the connection is not without interruption. The time needed for switching depends on the size and structure of the network.

The use of link aggregation which is often also called "trunking" likewise provides increased transmission reliability. Link aggregation actually means the parallel connection of several data lines. This way the bandwidths of the individual data lines are bundled in order to increase the total bandwidth. Furthermore, the parallel connection establishes a redundant connection. If one data line fails, the data can still be transmitted via the remaining lines even though only with reduced bandwidth.

3.2.3 Safe models

To obtain a certain grade of safety for the transmitted data against unauthorized access or to optimize the network utilization, it is suitable to design so-called Virtual Bridged Local Area Networks (VLANs). In a VLAN the data flow is grouped. The simplest variant of a VLAN is obtained by a port-related grouping which means that particular ports of a switch are assigned to a VLAN and data exchange is then only performed within this VLAN. A VLAN can be considered as a group of terminal stations which communicate like in a usual LAN although they can be located in different physical segments. In the end, establishing VLANs leads to a limitation of the broadcast domains. As a result, all subscribers of a VLAN only receive data packages which have been sent by subscribers of the same VLAN. Independent of their physical location, all subscribers of a VLAN are logically put together to one broadcast domain. The limitation of the broadcast domains relieves load from the network and provides safety since only the members of the VLAN are able to receive the data packages.

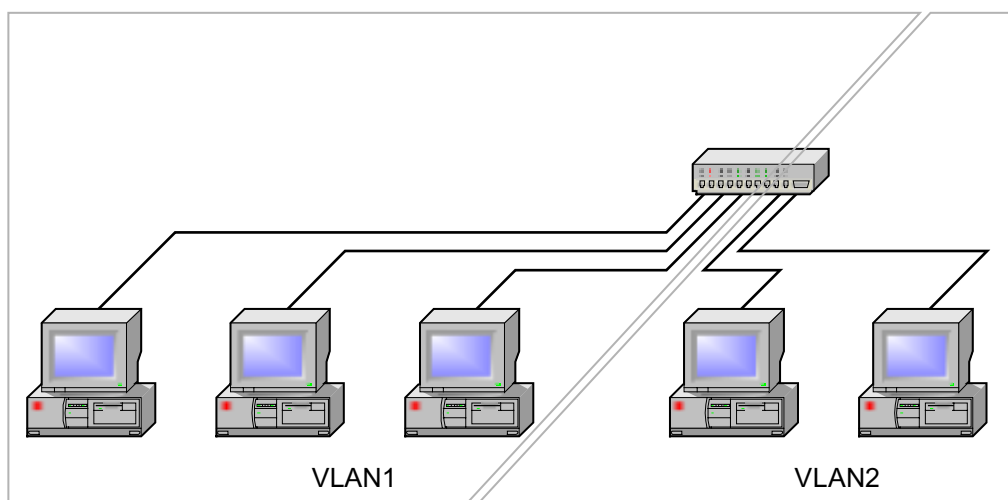


Figure 3-3 : Safe models

If a terminal device which is connected to a switch shall be able to exchange data beyond the borders of the VLAN, the port of the switch has to be assigned to several VLANs. Apart from the simple variant of the port-based VLAN it is also possible to establish VLANs by evaluating additional information contained in the Ethernet frames.

3.3 Utilization and performance

In the description of the network models it has already been mentioned that the existing hubs should be replaced by switches in order to subdivide and thus relieve the collision domain, if the utilization of a shared media is higher than 40 % over a longer period of time. If the utilization within such a switched media is permanently above 80 %, it is recommended to further relieve load by performing smaller grouping.

However, a network should basically not be dimensioned for the burst utilization. During normal operation usually many smaller data packages are transmitted rather than large data streams. This means that the network load regarding the bandwidth is not as high. Nevertheless, if any bottle-necks occur, the simplest method to eliminate them is to increase the data rate (e.g. from 10 Mbit/s to 100 Mbit/s). In existing networks, however, this is not always possible without problems since the cable infrastructure is possibly not suitable for the higher data rate and the expenditure for a new cabling is possibly not defensible. The only solution in such cases is a segmentation of the network which results in a reduction of the number of devices within the network or collision domain and thus provides more bandwidth for the remaining devices.

A segmentation of a network can be obtained with routers, bridges or switches. However, a segmentation is only meaningful if the 80/20 rule is considered and observed. The 80/20 rule says that 80 % of the data traffic have to take place within the segment and only 20 % of the data traffic are forwarded to another segment. This is why a previous analysis of the network traffic is required to enable meaningful grouping. In this analysis it has to be determined which station is communicating with which other stations in the network and which amount of data is flowing for this communication. For shared media the network should be divided in a way that stations producing roughly the same load should be grouped in one collision domain, if it is not possible to make a division based on the communication paths. This way it is guaranteed that stations with lower data traffic are able to meet the typical requirements regarding short response times. Stations with permanently high data traffic generally cause a drastic increase of the response times.

Best performance increase can be obtained by using switches and connecting each single station directly to the switches. This way each station has its own connection to a switch and thus can use the full bandwidth of a port in full-duplex mode. This subdivision and the provision of the dedicated connections is called **micro-segmentation**. For micro-segmentation the 80/20 rule does no longer apply. It has only to be guaranteed that a switch is able to provide sufficient internal bandwidth.

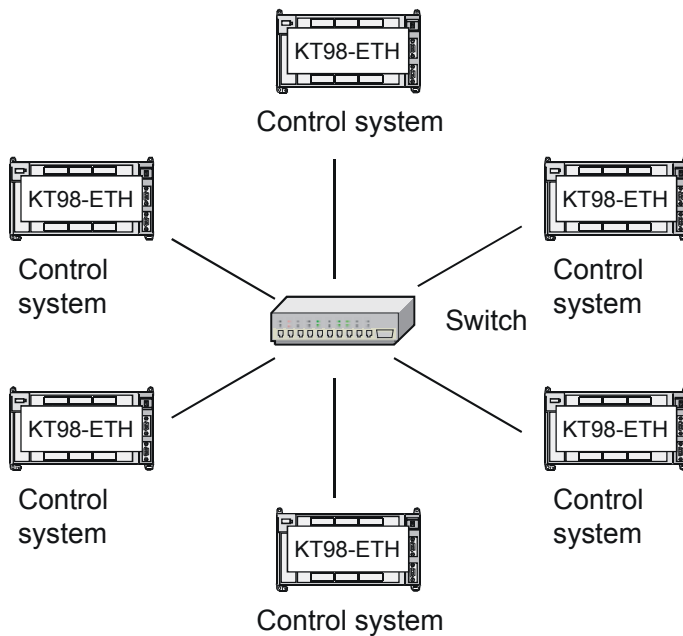


Figure 3-4 : Direct connection of all stations to switches

In order to plan a network with optimum performance, we have to think about the question what a network is able to achieve at all. Taking the standards as a basis it can be determined how many data per time can be transmitted via a network theoretically. The smallest Ethernet frame size is 64 bytes long and contains 46 bytes of user data, the maximum frame size is 1518 bytes at 1500 bytes of user data, each plus 64 bits for the preamble and 96 bits for the inter-frame gap. This results in a minimum length of 672 bits ($64 \times 8 + 64 + 96$) and a maximum length of 12304 bits ($1518 \times 8 + 64 + 96$). The transmission of one bit takes 10 ns for fast Ethernet (100 Mbit/s) and 100 ns for Ethernet (10 Mbit/s). Using these values we can calculate how many data packages of the smallest and the maximum length can be transmitted per second theoretically (see tables). The calculation of the corresponding amount of user data which can be transmitted (without taking into account the additional overheads of the higher protocols) now shows the considerably higher protocol overhead caused by the small data packages.

10 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/frame [1/s]	User data [bytes/s]
min. frame	672	100	67,200	14,880	46	684,480
max. frame	12,304	100	1,230,400	813	1,500	1,219,500

Table 3-1 : Data rate at 10 Mbit/s

100 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/frame [1/s]	User data [bytes/s]
min. frame	672	10	6,720	148,800	46	6,844,800
max. frame	12,304	10	123,400	8,127	1,500	12,195,000

Table 3-2 : Data rate at 100 Mbit/s

The corresponding net bandwidth can be calculated from the ratio of the amount of user data per second to the available network bandwidth. The net bandwidth is independent of the transmission rate and calculated in the following table taking a transmission rate of 100 Mbit/s as an example.

100 Mbit/s	User data [bit/s]	Network bandwidth [bit/s]	Net bandwidth [%]
min. frame	54,758,400	100,000,000	54.7
max. frame	97,524,000	100,000,000	97.5

Table 3-3 : Net bandwidth at 100 Mbit/s

These calculations point out that the percentage of the network performance is considerably higher for the transmission of larger frames. The efficiency of the data transmission which is independent of the transmission rate is shown in the following table using some selected frame sizes as an example. However, the values given in the table only consider the protocol overhead of the MAC and the network layer. The user data are reduced accordingly by the additional overhead of the corresponding higher layers.

User data [bits]	Frame size [bits]	Overhead [%]	Efficiency [%]
1500	1518	1.2	98.8
982	1000	1.8	98.2
494	512	3.6	96.4
46	64	39.1	60.9

Table 3-4: Efficiency of data transmission

A calculation of the typical transmitted frame sizes may be still possible for small closed networks inside an installation with only automation devices connected. But, for instance, if PCs are additionally connected to the network (even if they are connected only temporarily) the frame sizes can vary considerably. This makes it impossible to perform an exact calculation of the bandwidth or to make a precise statement regarding the performance. However, the following index values could be determined with the help of various studies about network performance.

- For low utilization of 0 to 50 % of the available bandwidth, short response times can be expected. The stations are able to send frames with a typical delay of smaller than 1 ms
- For medium utilization between 50 and 80 % the response times can possibly increase to values between 10 and 100 ms.
- For high utilization over 80 % high response time and wide distribution can be expected. The sending of frames can possibly take up to 10 seconds.

This is why the following principles should be obtained when designing an Ethernet network.

- Mixed operation of stations which have to transmit high data volumes and stations which have to operate with short response times (real time) should be avoided. Due to the wide distribution, short response times cannot be guaranteed within such combinations.
- As few as possible stations should be positioned inside of one collision domain. For this purpose, collision domains should be subdivided using switching hubs.

4 Design examples

4.1 Introduction


The Ethernet coupler is only completely ready for operation if a valid configuration has been loaded before. Configuring the coupler is always necessary, independent of the intended use of the coupler. Only the parameters to be adjusted depend on the application. Furthermore, some cases require additional implementation within the user program.

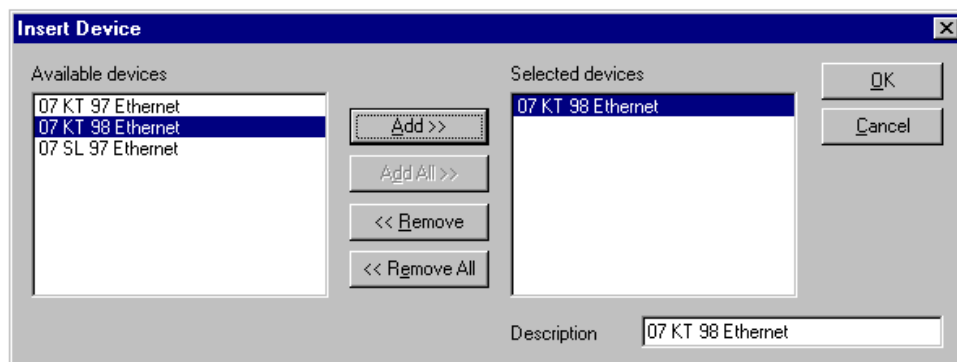
In the following sections the general procedure for configuring the Ethernet coupler and the application-specific parameter assignment are explained as well as the possibly required implementation in the user program. In each section regarding the application-specific parameter assignment, one specific application case is described separately. Of course the different functions can also be mixed in any combination.

4.2 General procedure for configuring the coupler

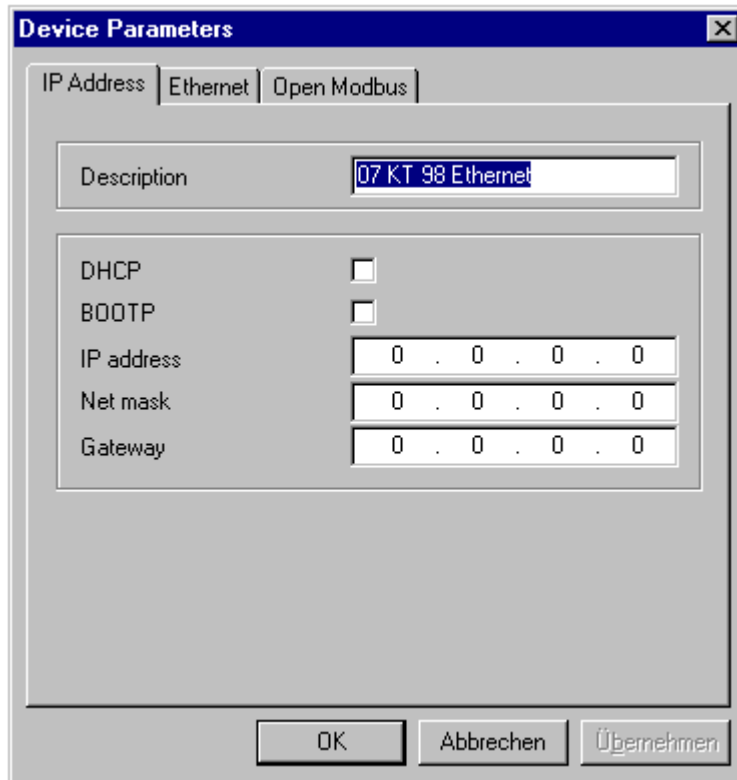
In this section the general procedure for configuring the coupler is described. This procedure has to be performed in any case. The further application-specific parameter assignment is described in the subsequent sections.

To create a configuration file, you first have to start the field bus configuration tool **907 FB 1131**. Select **File | New** from the menu bar. In the appearing dialog box select **Ethernet / Protocol** and then confirm with **OK**.

Now insert a device using the button . After left-clicking on this button a selection dialog listing all available controllers with Ethernet couplers appears. Highlight the controller type you are using and apply your selection with **Add>>**. In the **Description** text field enter any name for the device and confirm with **OK**.



The selected controller is then displayed in the main screen. Setting the configuration parameters can now be done by double clicking on the device or via the menu item **Settings | Device Parameters**.



In this dialog perform your application-specific settings as described in the following subsections. Confirm your entries with **OK**.

Save the finished configuration file.

Finally, the configuration file has to be downloaded to the corresponding Ethernet coupler. For this purpose, use the menu item **Settings | Device Assignment** to select the access for downloading the file to the coupler. Two drivers are available for selection:

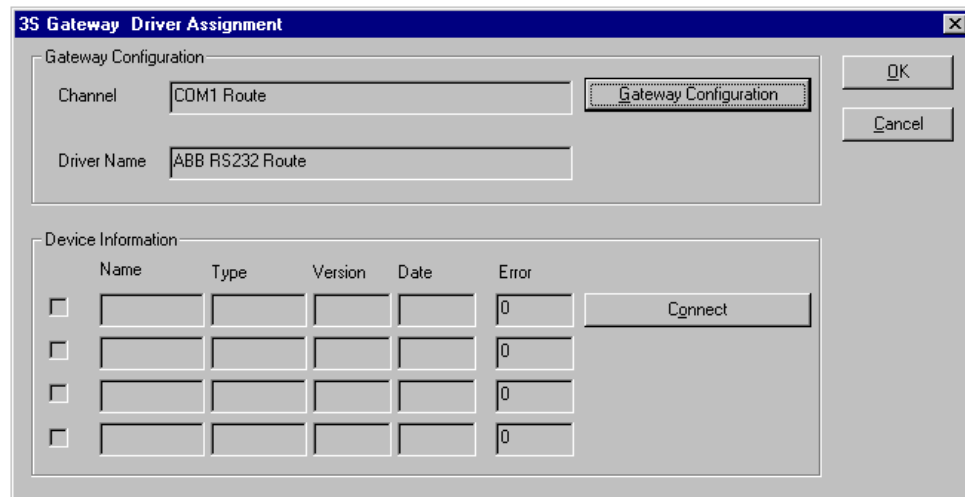
- Gateway driver
- CIF TCP/IP driver

The gateway driver allows to download the configuration via all known ways of access except Ethernet. To download the configuration directly via Ethernet TCP/IP from the PC to the controller, you have to use the CIF TCP/IP driver.

Select the desired driver and confirm with **OK**.

Gateway driver:

The dialog for assigning the gateway driver is opened:



Click on the **Gateway Configuration** button to open an existing gateway channel or to specify a new channel. The dialog for configuring the gateway appears (refer to System technology of the basic units / Programming and testing).

Remark:

Generally all offered drivers are available for setting the gateway channel. However, for the TCP/IP driver it has to be observed that this driver can be used as programming and diagnosis access (online access). It cannot be used to download the configuration data. To download the configuration directly via Ethernet TCP/IP from the PC to the controller, you have to use the CIF TCP/IP driver.

Set the gateway channel parameters appropriately according to the controller to which you want to download the data. After setting and selecting the desired channel, 907 FB 1131 automatically tries to establish a connection to the controller. You can also repeat this procedure manually by selecting **Connect**.

If the connection has been established successfully, various information concerning the couplers detected in the controller are displayed in the **Device Information** area and the corresponding **Error** fields contains the value 0. If no communication could be established, the **Error** field contains a numerical code specifying the occurred error (refer to the documentation for the field bus configuration tool 907 FB 1131). After the connection has been established successfully, you have to confirm the selection of the concerning coupler again by **activating the relevant checkbox** and then clicking on **OK**. Now the selected channel is uniquely assigned to the opened configuration file. Select the menu item **Online | Download** and confirm the subsequent security dialog asking you whether you really want to perform the download with **Yes**. A window appears displaying the progress of the download. The window is automatically closed after the download is completed.

Now the coupler is configured and completely ready for operation.

CIF TCP/IP driver:

The device assignment dialog for the TCP/IP driver is opened:

Driver Description

Driver: ODMTcplp V2.014

Add IP Address

IP Address: [] Add

Board Selection

	IP Address	Type	Serial Number	MAC Address	Address Switch
<input type="checkbox"/>	10.49.91.250	C104-EN	11	00-02-A2-14-00-0B	0xF0
<input type="checkbox"/>	10.49.91.252	C104-EN	9	00-02-A2-14-00-09	0xF2
<input checked="" type="checkbox"/>	10.49.91.251	C104-EN	13	00-02-A2-14-00-0D	0xF1
<input type="checkbox"/>	10.49.91.254	C104-EN	47	00-02-A2-14-00-2F	0xF4
<input type="checkbox"/>	10.49.91.253	C104-EN	83	00-02-A2-14-00-53	0xF3

Filtered Device(s)

IP Address	Type	Serial Number	MAC Address	Address Switch

Select device

Buttons: OK, Cancel, NetIdent Rescan, Set IP Address

First, the driver automatically searches for series 90 controllers with Ethernet couplers that can be accessed via the network. The found devices are displayed in the **Board Selection** list. You can also start the scanning process manually by clicking on the **NetIdent Rescan** button.

Remark:

Every Ethernet interface can be uniquely identified by its MAC address. But normally the MAC address is not visible on the device from the outside. Thus, no physical assignment of a MAC address to a particular device can be performed. Therefore the address switch status of the found controllers is additionally displayed beneath the MAC address in order to ease the assignment of a particular device. This way up to 256 controllers can be easily identified within a network. The address switches are located at the outside of the housing, immediately next to the Ethernet connector. Identifying a controller by means of the address switches is particularly advantageous if the concerning Ethernet coupler does not yet have a valid configuration (IP address 0.0.0.0, e.g. on delivery).

Prior to a configuration download a valid IP address has to be assigned to the desired controller. For this purpose, select the corresponding controller by **checking the checkbox** in the concerning line with a hook and then clicking on the **Set IP Address** button. After entering the IP address and activating the assignment process you are informed whether the IP address could be assigned successfully.

Now the configuration download can be done. Select the menu item **Online | Download** and confirm the subsequent security dialog asking you whether you really want to perform the download with **Yes**. If the coupler does already have a valid IP address which is different from the IP address set in the configuration file, you are additionally requested to specify which of the

two addresses you want to use. A window appears displaying the progress of the download. The window is automatically closed after the download is completed.

Now the coupler is configured and completely ready for operation.

4.3 Ethernet used for online access

The simplest application for the Ethernet coupler is to use it for online access exclusively. This access can be used by all programs accessing a controller via the gateway, e.g. 907 AC 1131, 907 FB 1131, OPC and DDE.

4.3.1 Configuring the coupler

First create a configuration file as described in section 4.2 "General procedure for configuring the coupler" and specify the device parameters. If the Ethernet coupler is exclusively used for online access, only the parameters in the tabs **IP Address** and **Ethernet** have to be specified. Since Modbus on TCP/IP is not used, the default values of the parameters listed in the **Open Modbus** tab can be left unchanged.

All IP protocol-specific parameters are put together in the **IP Address** parameter group. Here it has to be specified first whether the IP address shall be assigned via a BOOTP and/or DHCP server. If the IP address is assigned via a server it has to be observed that the controller cannot be called via TCP/IP until it has been provided with an IP address by a corresponding server. This assignment process is repeated with each boot process of the controller and can take up to three and a half minutes. Depending on the assignment mechanism used by the server, the IP address of the coupler can furthermore change with each boot process of the controller. Then the currently valid IP address first has to be determined again after each boot process by means of NetIdent and an evaluation of the address switches. IP address assignment via a BOOTP or DHCP server is usually not used for pure automation systems. For information about this please contact your system administrator.

In the fields **IP address** and **Net mask** the IP address and the net mask have to be specified the Ethernet coupler shall use (like for a PC). These inputs are mandatory if neither **BOOTP** nor **DHCP** is used. In all other cases the inputs in these fields are only used if the assignment by the server(s) fails.

Specifying the IP address of a (standard) **Gateway** is optional and only required if datagrams shall be transmitted to a device located outside of the local network.

Please contact your system administrator for information about all parameters to be set in this tab.

Starting from the assumption that the controller is used in a closed installation-internal network without a BOOTP and a DHCP server the following example parameter settings result.

Device Parameters	
IP Address	Ethernet Open Modbus
Description	07 KT 98 Ethernet
DHCP	<input type="checkbox"/>
BOOTP	<input type="checkbox"/>
IP address	10 . 49 . 91 . 251
Net mask	255 . 255 . 255 . 0
Gateway	0 . 0 . 0 . 0
OK Abbrechen Übernehmen	

The **Ethernet** parameter group includes all Ethernet-specific parameters. In this tab you can first specify the Ethernet interface (Twisted Pair or AUI) to be used or select whether you want to have the used interface detected automatically. As Twisted Pair is the only supported interface at the moment, it is recommended to leave the default values unchanged.

The standardized **Auto negotiation** functionality offers the option that all stations in the network automatically agree on the highest possible transmission rate and, if possible, full duplex operation. If the auto negotiation function is selected it has to be guaranteed that all devices in the network are set to this mechanism.

Instead of using the auto negotiation function, the **duplex mode** (half or full) and the transmission **speed** (10 Mbit/s or 100 Mbit/s) can be preset as required. In this case it has also to be guaranteed that all subscribers in the network have the same setting. Otherwise no communication can be established.

If a hub is located between two stations on a transmission line, the devices can be operated only in half-duplex mode. This is automatically guaranteed if the auto negotiation function is used. Otherwise this setting has to be done manually. Selecting full-duplex operation instead will cause communication errors. Full-duplex operation is only possible if a network is completely set up with switches.

The **MAC address** is the world-wide unique hardware address of the Ethernet chip. This address is normally burned into the chip during manufacturing. However, for special installations it can be necessary for an Ethernet subscriber to have a specific MAC address. This is why it is possible to predetermine the MAC address of an Ethernet coupler by configuration. In this case the coupler does not answer with its burned-in MAC address but with the configured MAC address. For this purpose, select **Set MAC address** and enter the desired MAC address in the form of six hexadecimal coded bytes.



Warning:

Normally it is not necessary to change the burned-in MAC address of an Ethernet coupler. This burned-in MAC address is world-wide unique. Changing the address can damage the function of a network, if the network then contains devices with an identical MAC address.

Since the installation used in this example consists of several controllers and a PC connected to each other via Twisted Pair cabling and a switch, the following Ethernet parameters result:

The screenshot shows a dialog box titled "Device Parameters" with three tabs: "IP Address", "Ethernet", and "Open Modbus". The "Ethernet" tab is active. It contains several configuration options:

- Description: 07 KT 98 Ethernet
- Auto detect:
- Interface: Twisted Pair (dropdown menu)
- Auto negotiation:
- Duplex mode: Half (dropdown menu)
- Speed: 10 MBit/s (dropdown menu)
- Set MAC address:
- MAC address (hex): FF - FF - FF - FF - FF - FF

At the bottom of the dialog are three buttons: "OK", "Abbrechen", and "Übernehmen".

The application-specific adjustment of the device parameters is now completed. Continue as described under 4.2 "General procedure for configuring the coupler" and download the configuration data to the Ethernet coupler.

4.4 Modbus on TCP/IP

For the standardized (Open)Modbus protocol two operating modes are distinguished. Controllers with Ethernet coupler can be operated as Modbus on TCP/IP client (master) as well as as server (slave). Simultaneous operation as client and server is also possible. In all operating modes several Modbus on TCP/IP connections can be provided simultaneously. The maximum possible number of simultaneous connections is only limited by the number of available sockets. Here it has to be observed that each additional communication connection aside from Modbus on TCP/IP also requires one or more sockets.

Operation of the controller as Modbus server only requires a corresponding configuration for the Ethernet coupler to be set up. Operation as Modbus client furthermore requires the implementation of blocks in the user program.



Warning:

The user program should not be implemented as a cyclic task (PLC_PRG) if high communication traffic is expected due to the operation as Modbus on TCP/IP client and/or server. A task configuration has to be implemented instead with the cycle time to be adjusted in a way that sufficient communication resources are available in addition to the actual processing time for the program.

4.4.1 Operation as server / slave

A typical application for the operation of a controller as (Open)Modbus on TCP/IP server is the linking of an operating terminal via Ethernet. With this application the operating terminal operates as Modbus client and sends telegrams for reading or writing variables in the controller which executes them accordingly. Since (Open)Modbus on TCP/IP is a standardized protocol, every device supporting this protocol can be connected this way independent of the device type and manufacturer.

For our design example a closed installation-internal network is used. The network consists of four subscribers, two operating terminals and two controllers of the type 07 KT 98. The devices are connected to each other via a switch. The operating terminals shall have the IP addresses 10.49.91.251 and 10.49.91.252. The controllers shall have the IP addresses 10.49.91.253 and 10.49.91.254. Both operating terminals shall be able to access both controllers. Since the terminals are not time-synchronized, simultaneous access of both terminals to the same controller must be possible.

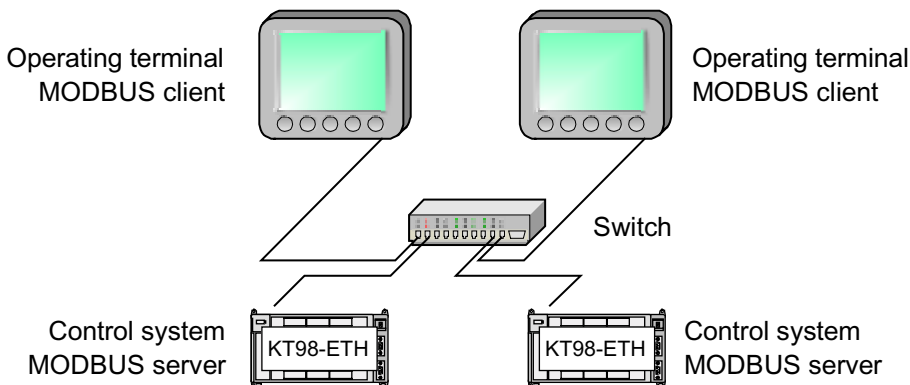


Figure 4-1 : Example for OpenModbus on TCP/IP - controller operated as server

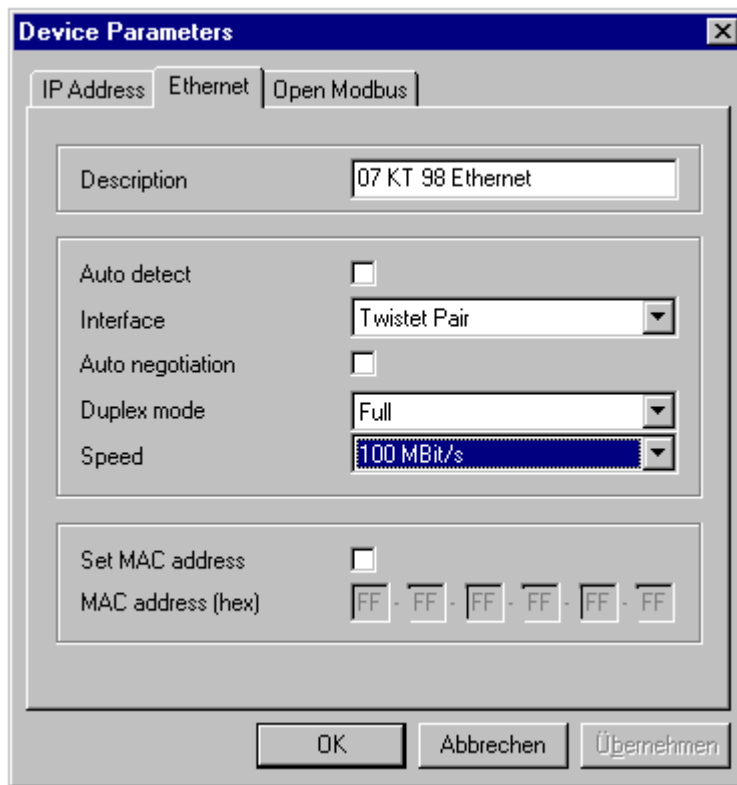
Designing the operating terminals as Modbus clients is not subject of this description. For information about this please contact the corresponding manufacturer. Please refer to the description of the MODMAST function block contained in the 907 AC 1131 documentation for the cross-reference list for client access to the operands of the controller.

Configuring the coupler:

In order to design the first controller, first create a configuration file as described in section 4.2 "General procedure for configuring the coupler" and specify the device parameters. Since the IP addresses are predetermined for all devices, enter in the **IP address** parameter group the value 10.49.91.253 for the **IP address** and 255.255.255.0 for the **Net mask**. Leave the checkboxes **BOOTP** and **DHCP** blank since neither a BOOTP server nor a DHCP server is used. It is furthermore not necessary to specify a **Gateway** IP address since the network does not have any connection to the outside world. Thus, the parameter settings for the first controller are as follows:

Device Parameters	
IP Address	Ethernet
Open Modbus	
Description	07 KT 98 Ethernet
DHCP	<input type="checkbox"/>
BOOTP	<input type="checkbox"/>
IP address	10 . 49 . 91 . 253
Net mask	255 . 255 . 255 . 0
Gateway	0 . 0 . 0 . 0
OK Abbrechen Übernehmen	

Switch to the **Ethernet** parameters group. For the interface select **Twisted Pair**. If the operating terminals support the **Auto negotiation** function and if this function is activated in the terminals, switch on the auto negotiation function by checking the corresponding checkbox. Then all devices automatically agree during operation on the transmission parameters providing optimum performance. In this example, however, the parameters shall be preset manually. Therefore, set the transmission rate to the same data rate as set at the operating terminal (here: 100 Mbit/s). Since all devices are connected to each other via one single switch, the network can furthermore operate in **full-duplex** mode. This parameter setting also has to be identical for all subscribers in the network. Thus, the Ethernet parameter settings are as follows:



Switch to the **Open Modbus** parameters group. This view contains all parameters necessary for operating the controller as Modbus client and/or server via TCP/IP.

For this example both operating terminals shall be able to access the controller simultaneously via Modbus. This is achieved by entering the value 2 for the **Server connections** which reserves two sockets for requests of Modbus clients.



Warning:

The specified number of parallel server connections results in a permanent reservation of a corresponding number of sockets for these connections. These sockets cannot be used by other protocols. If many clients are used which shall be able to access the controller simultaneously this can result in a lack of sockets for other protocols. In this case the maximum possible number of simultaneous server connections has to be reduced.

The parameters **Telegram timeout** and **Connection remain open time** are only relevant when operating the controller as Modbus client (refer to corresponding example) and can be left at the default values here.

The three TCP parameters **Send timeout**, **Connect timeout** and **Close timeout** are related to the TCP protocol used by Modbus. They are considered for client operation as well as for server operation. In the corresponding input fields the respective times have to be specified in milliseconds where 0 is the respective default timeout value. Valid values are 0 to 2.000.000.000.

The **Send timeout** parameter determines how long the controller shall attempt to return the answer to a client's request to the client. Normally the default value 0 which corresponds to 31 seconds can be kept.

The **Connect timeout** parameter determines how long the controller shall attempt to establish a TCP connection. Normally the default value 0 which corresponds to 31 seconds can be kept.

The **Close timeout** parameter determines how long the controller shall attempt to close a TCP connection. Normally the default value 0 which corresponds to 13 seconds can be kept.

The **Swap** parameter is also relevant for both the client operation and the server operation. With this parameter you can determine whether the two bytes in the words shall be swapped automatically during the transmission of data words. Swapping the word data can be necessary if devices with different processor types are used. If, for example, a client is sending word data in Motorola format, these data have to be converted into the Intel format (Swap = TRUE) prior to their transmission into the operand memory of the controller. In the opposite direction the word data in the operand memory of the controller are converted into the Motorola format prior to their transmission to the client.

Since the operating terminals used in this example are processing the data in Intel format, the Swap parameter has to be set to FALSE.

Thus, the Open Modbus parameter settings are as follows:

The screenshot shows the 'Device Parameters' dialog box with the 'Open Modbus' tab selected. The 'Description' field is filled with '07 KT 98 Ethernet'. In the 'Server connections' section, 'Server connections' is set to 2, 'Telegram timeout' is 20 * 100 ms, and 'Connect. remain open time' is 10 * 100 ms. The 'TCP' section has 'Send timeout', 'Connect timeout', and 'Close timeout' all set to 0. The 'Swap' dropdown menu is set to 'FALSE'. At the bottom, there are three buttons: 'OK', 'Abbrechen', and 'Übernehmen'.

The creation of the configuration data for the first controller is now completed. Continue as described under 4.2 "General procedure for configuring the coupler" and download the configuration data to the Ethernet coupler.

Repeat the configuration for the second controller using the IP address 10.49.91.254. All other parameter settings are identical to the settings for the first controller.

Implementation in the user program:

The design process for Modbus server operation is now completed. Since the protocol is automatically processed by the controller, no further blocks in the user program are required. If necessary, the function block **ETH_MODSTAT** can be inserted into the project for diagnosis. The function block is contained in the library **Ethernet_S90_V50.lib** (or higher version). In this case you have to insert the library **Coupler_S90_V50.lib** (or higher version) into your project in addition to the Ethernet library.

4.4.2 Operation as client / master

One possible application for the operation of a controller as (Open)Modbus on TCP/IP client is for example the linking of sensors with Ethernet connection supporting the Modbus protocol. With this application the controller operates as Modbus client and sends telegrams for reading or writing data to the corresponding sensor which in turn generates a corresponding answer. Since (Open)Modbus on TCP/IP is a standardized protocol, every device supporting this protocol can be connected this way independent of the device type and manufacturer.

For our design example a closed installation-internal network is used. The network consists of three subscribers, one controller of the type 07 KT 98 and two temperature sensors with Ethernet connection. The devices are connected to each other via a switch. The controller shall have the IP address 10.49.91.253. The sensors shall have the IP addresses 10.49.91.251 and 10.49.91.252.

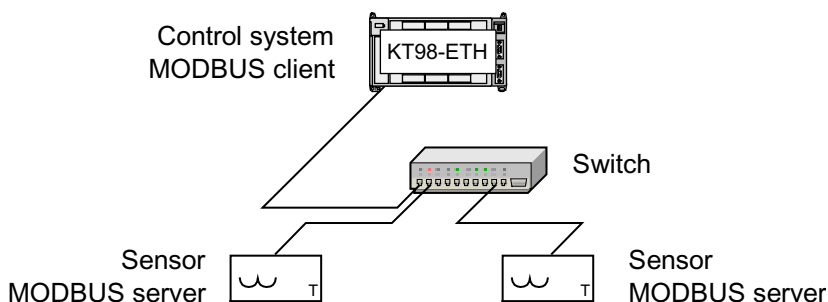
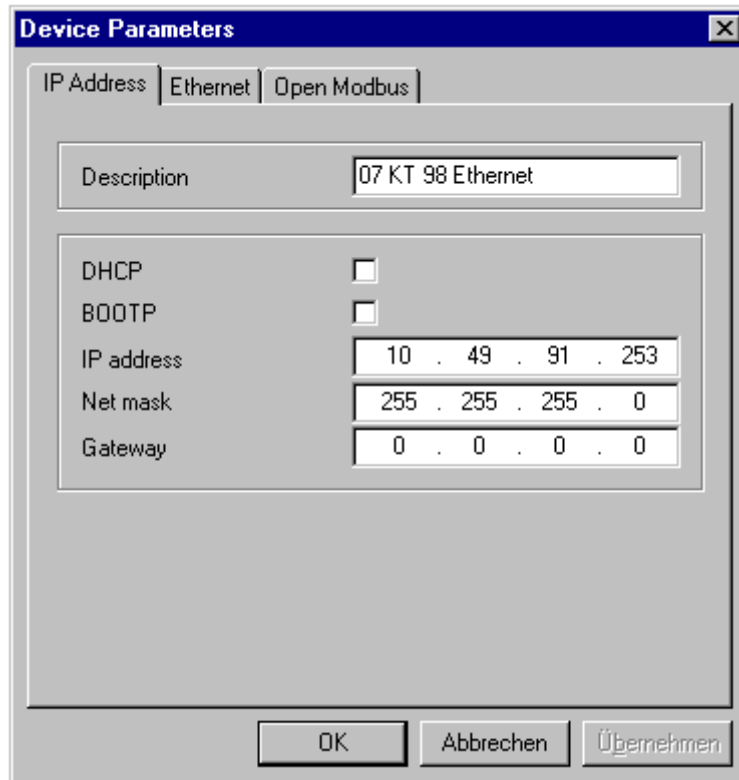


Figure 4-2 : Example for OpenModbus on TCP/IP - controller operated as client

Planning the sensors (setting the IP address) as Modbus server is not subject of this description. For information about this please contact the corresponding manufacturer.

Configuring the coupler:

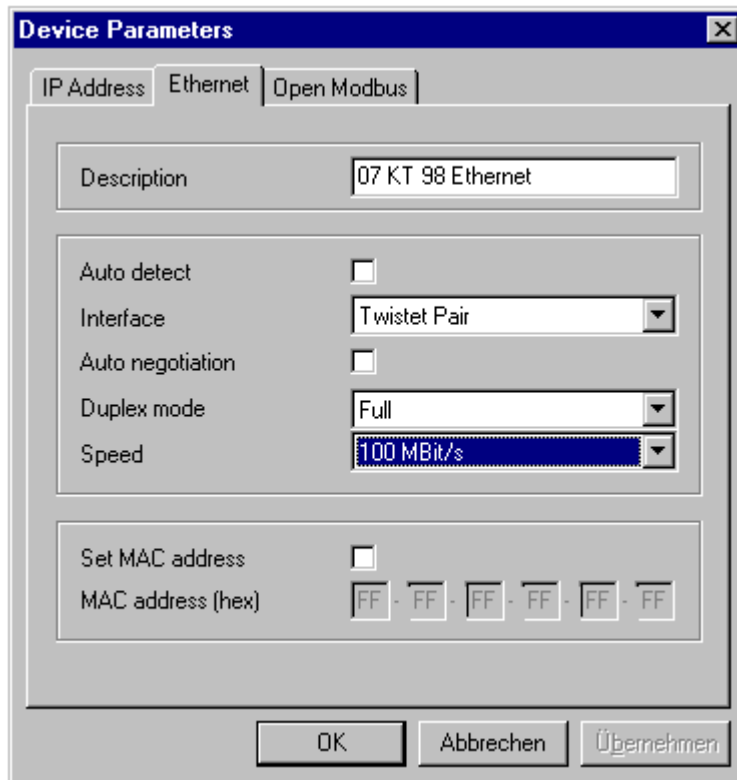
In order to design the controller, first create a configuration file as described in section 4.2 "General procedure for configuring the coupler" and specify the device parameters. Since the IP addresses are predetermined for all devices, enter in the **IP address** parameter group the value 10.49.91.253 for the IP address and 255.255.255.0 for the **Net mask**. Leave the checkboxes **BOOTP** and **DHCP** blank since neither a BOOTP server nor a DHCP server is used. It is furthermore not necessary to specify a **Gateway** IP address since the network does not have any connection to the outside world. Thus, the parameter settings for the first controller are as follows:



The screenshot shows a dialog box titled "Device Parameters" with three tabs: "IP Address", "Ethernet", and "Open Modbus". The "Ethernet" tab is selected. The "Description" field contains "07 KT 98 Ethernet". Below this, there are checkboxes for "DHCP" and "BOOTP", both of which are unchecked. The "IP address" field is set to "10 . 49 . 91 . 253", the "Net mask" field is set to "255 . 255 . 255 . 0", and the "Gateway" field is set to "0 . 0 . 0 . 0". At the bottom of the dialog, there are three buttons: "OK", "Abbrechen", and "Übernehmen".

Parameter	Value
Description	07 KT 98 Ethernet
DHCP	<input type="checkbox"/>
BOOTP	<input type="checkbox"/>
IP address	10 . 49 . 91 . 253
Net mask	255 . 255 . 255 . 0
Gateway	0 . 0 . 0 . 0

Switch to the **Ethernet** parameters group. For the interface select **Twisted Pair**. If the sensors support the **Auto negotiation** function and if this function is activated in the sensors, switch on the auto negotiation function by checking the corresponding checkbox. Then all devices automatically agree during operation on the transmission parameters providing optimum performance. In this example, however, the parameters shall be preset manually. Therefore, set the transmission rate (**Speed**) to the same data rate as set at the sensors (here: 100 Mbit/s). Since all devices are connected to each other via one single switch, the network can furthermore operate in **full-duplex** mode. This parameter setting also has to be identical for all subscribers in the network. Thus, the Ethernet parameter settings are as follows:



Switch to the **Open Modbus** parameters group. This view contains all parameters necessary for operating the controller as Modbus client and/or server via TCP/IP.

For this example the controller shall be used exclusively as Modbus client. Consequently enter 0 for the **Server connections** since no server connections are required. In contrast to server operation where the number of sockets specified for "Server connections" is permanently seized, in client operation the required sockets are dynamically requested and released again as required.

The parameters **Telegram timeout** and **Connection remain open time** are relevant for controller operation as Modbus client.

The **Telegram timeout** parameter determines how long the client shall wait for the server's answer after the transmission of the request to the server before the process is aborted with an error message. For the network considered in our example the default value of 20 x 100 ms can be maintained or even reduced. However, for larger networks with high utilization it can be necessary to increase this value. The valid values for the telegram timeout parameter range from 1 x 100 ms to 60000 x 100 ms. Here it has to be observed that the controller normally cannot access a server either until it has received its answer or until the timeout has expired. This is why the telegram timeout value should be dimensioned in a way that no abortion occurs even during temporarily higher utilization. On the other hand, the time reserve should not be too high in order to enable the detection of communication errors as early as possible for a

correspondingly high connection performance. For this example a telegram timeout value of 5 x 100 ms is selected.

The **Connection remain open time** specifies how long the connection shall be maintained after reception of the response telegram. Modbus is based on TCP/IP. One characteristic of this protocol is that a logical communication connection is first established, then the data are exchanged and finally the connection is closed again. The process for establishing and closing the connection takes some time. If data communication between the controller and the servers shall only be performed in longer intervals it makes sense to close the connection immediately after data communication is finished. This avoids unnecessary long blocking of the Modbus access. For fast cyclic data exchange between the client and the server the frequency of connection establishment and closing can be reduced by setting a higher "connection remain open time".



Warning:

It has to be observed that the concerning Modbus access (socket) is blocked for other clients as long as the connection is established. If a Modbus server only has a logical access, this results in the fact that no other client can access this server during this time. As long as the connection is established, it also seizes a socket on the controller side. Under certain circumstances this can lead to the fact that not enough sockets can be made available for other protocols during this time.

For this example it is committed that the controller operating as Modbus client shall request the temperature every second from the sensors operating as Modbus servers. Since no other client is accessing the sensors, the connections can be kept open for 1.5 seconds. The remain open time is restarted for each connection after each reception of a response from the corresponding server. Therefore, please enter the value 15 x 100 ms here.

The three TCP parameters *Send timeout*, *Connect timeout* and *Close timeout* are related to the TCP protocol used by Modbus. They are considered for client operation as well as for server operation. In the corresponding input fields the respective times have to be specified in milliseconds where 0 is the respective default timeout value. Valid values are 0 to 2.000.000.000.

The **Send timeout** parameter determines how long the controller shall attempt to send a request to a server. Normally the default value 0 which corresponds to 31 seconds can be kept.

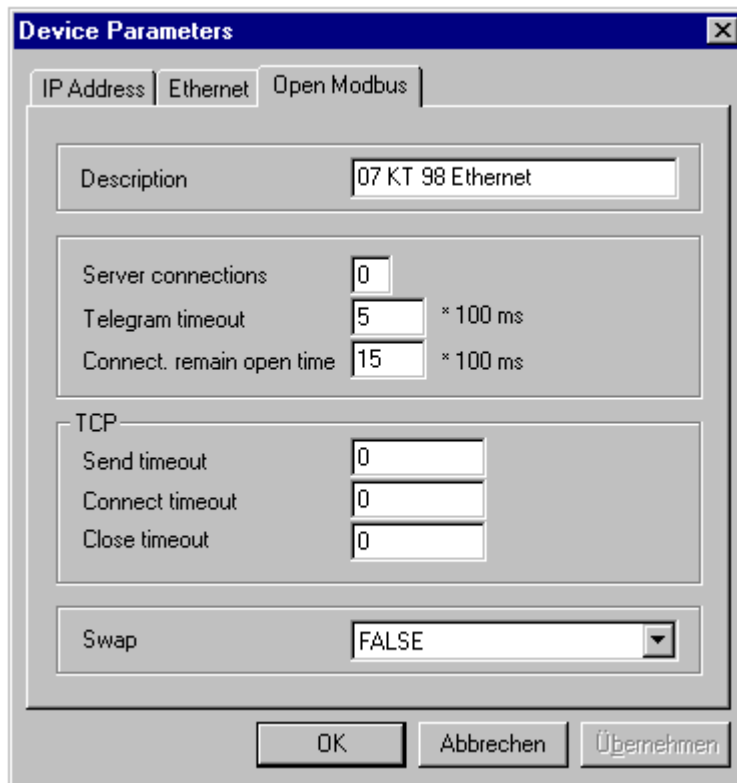
The **Connect timeout** parameter determines how long the controller shall attempt to establish a TCP connection to a server. Normally the default value 0 which corresponds to 31 seconds can be kept.

The **Close timeout** parameter determines how long the controller shall attempt to close a TCP connection to a server. Normally the default value 0 which corresponds to 13 seconds can be kept.

The **Swap** parameter is also relevant for both the client operation and the server operation. With this parameter you can determine whether the two bytes in the words shall be swapped automatically during the transmission of data words. Swapping the word data can be necessary if devices with different processor types are used. If, for example, a server is operating in Motorola format, the word data from the operand memory sent by the controller have to be converted from Intel format into Motorola format (Swap = TRUE) prior to the actual transmission to the server. In the opposite direction the word data of the server are converted into Intel format before they are written to the operand memory of the controller.

Since the servers used in this example are processing the data in Intel format, the Swap parameter has to be set to FALSE.

Thus, the Open Modbus parameter settings are as follows:



The creation of the configuration data is now completed. Continue as described under 4.2 "General procedure for configuring the coupler" and download the configuration data to the Ethernet coupler.

Implementation in the user program:

The coupler is ready for Modbus client operation. Now the blocks for starting the requests to the servers have to be implemented into the user program.

For this purpose, start the programming software 907 AC 1131 and create a new project for a controller 07 KT 98 or open a corresponding existing project. First select **Window | Library Manager | Additional Library** to integrate the files **Coupler_S90_V50.lib** (or higher version) and **Ethernet_S90_V50.lib** (or higher version) into the project. Then insert an instance of the **ETH_MODMAST** block into your project.

For controllers with only one Ethernet coupler the coupler is located in the first slot. Therefore assign the value 1 to block input **CONO**.

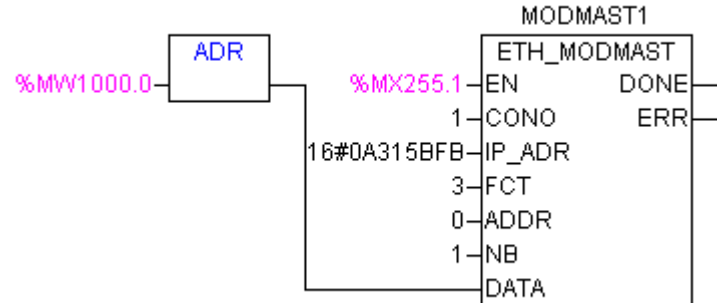
The IP address of the first sensor is 10.49.91.251. This IP address has to be applied to input **IP_ADR** of the MODMAST block. Each byte in IP_ADR represents one octet of the address. Thus, for our example the value 16#0A315BFB (hexadecimal) or 171006971 (decimal) has to be assigned to IP_ADR.

Since the temperatures shall be read from the sensors in the form of a word, the value 3 has to be assigned to **FCT** and 1 has to be specified for **NB**. At input **ADDR** the register address in the server has to be specified from which the word should be read. Starting from the assumption that the current temperature is stored in the sensor under register address 0, you have to enter the value 0 here.

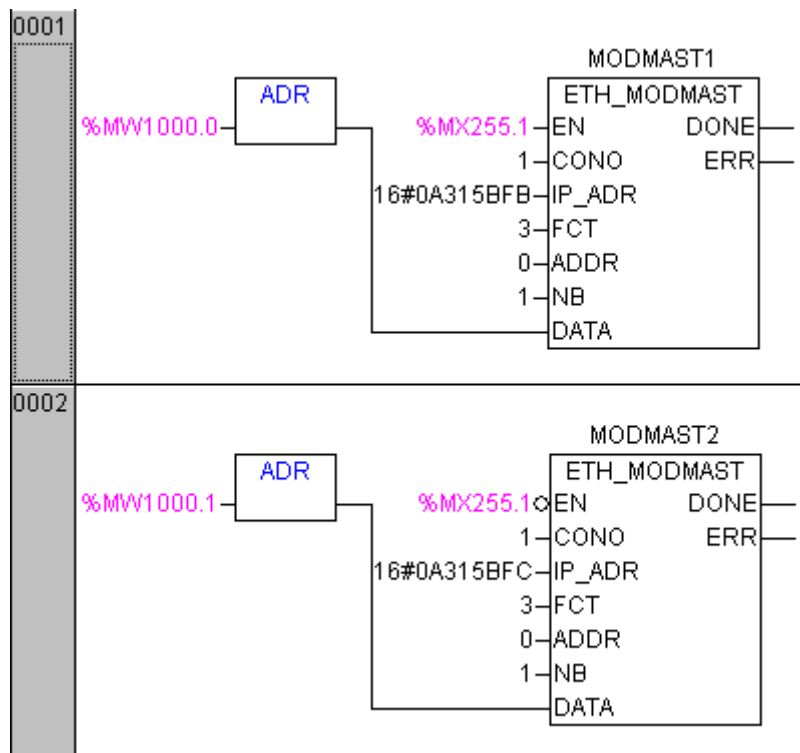
Furthermore the data read from the first sensor shall be stored in flag word %MW1000.0 in the operand memory of the controller. Thus, block input **DATA** has to be wired with this operand via the ADR block.

The Modbus request to a server is started with a FALSE → TRUE edge at input **EN**. Since the temperature shall be read every second, it is here suitable to assign flag %MX255.1 to the input.

Represented in FBD this results in the following program part for the first server.



Now insert a corresponding block for the second server. The assignments for the block inputs are almost identical. Only the IP address (16#0A315BFC) and the operand used to store the read values (%MW1000.1) are different. You can also negate input EN of the ETH_MODMAST block for the second server in order to avoid that the requests are initiated to both servers simultaneously. This causes the requests to be transmitted with a time-shift of 500 ms. So, the resulting program is as follows:



If required, the program additionally has to be expanded by an evaluation of possible errors. If an error occurs during the processing of a Modbus request, this is indicated at output **ERR** of the corresponding **ETH_MODMAST** block. The general Modbus processing status can be displayed and evaluated using the **ETH_MODSTAT** block.

Download the project to the controller. After this, implementation of the Modbus client is completed.

4.5 Fast data communication via UDP/IP

4.5.1 Example configuration for data communication via UDP/IP

Fast data communication via UDP/IP is a proprietary (manufacturer-specific) protocol. This protocol is only supported by AC31 series 90 controllers or newer ones. It serves for the transmission of any data between the controllers and can be used in parallel to all other protocols. This protocol is based on the standardized protocols UDP and IP. The actual protocol is lying above UDP/IP.

The protocol handling corresponds to a large extent to the new ARCNET processing. Aside from the corresponding configuration of the Ethernet coupler, the implementation of the following blocks is required for fast data communication via UDP/IP:

- ETH_AINIT
- ETH_AREC
- ETH_ASEND
- ETH_ASTO



Warning:

The user program should not be implemented as a cyclic task (PLC_PRG) if high communication traffic caused by the protocol for fast data communication via UDP/IP is expected. A task configuration has to be implemented instead with the cycle time to be adjusted in a way that sufficient communication resources are available in addition to the actual processing time for the program.

For our design example a closed installation-internal network is used. The network consists of three controllers of the type 07 KT 98. The devices are connected to each other via a switch. The controllers shall have the IP addresses 10.49.91.251, 10.49.91.252 and 10.49.91.253. Each controller shall send 8 bytes of data to each of the other two controllers. The uniform and constant user data length has been chosen only to simplify the example. If necessary, it is also possible to use different data lengths for each controller as well as variable data lengths for each transmission.

The individual transmissions shall be controlled in different ways for each controller.

Controller 1 (IP address 10.49.91.251)
transmits the data to the other two controllers periodically.

Controller 2 (IP address 10.49.91.252)
only transmits the data to the other two controllers if they have changed.

Controller 3 (IP address 10.49.91.253)
only transmits the data to one of the other two controllers if it has received data from this controller before.

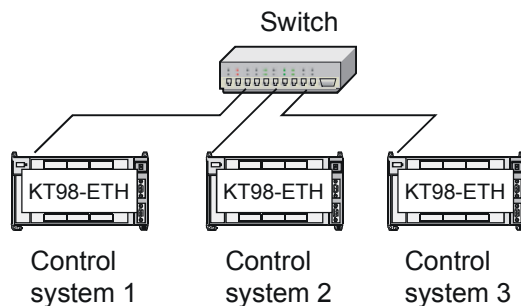


Figure 4-3 : Example configuration for data communication via UDP/IP

4.5.2 Configuring the Ethernet couplers for data communication via UDP/IP

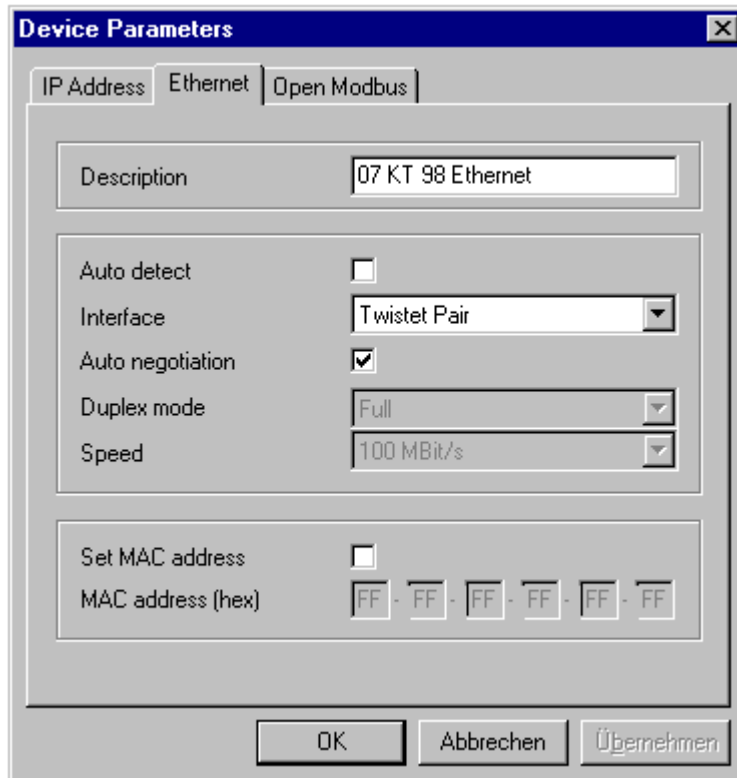
In order to configure the first controller, first create a configuration file as described in section 4.2 "General procedure for configuring the coupler" and specify the device parameters. The procedure and scope for configuring the individual device parameters are the same as for the operation of the Ethernet coupler exclusively for online access (see also section 4.3 "Ethernet used for online access").

Since the IP addresses are predetermined for all devices, enter in the **IP address** parameter group the value 10.49.91.251 for the **IP address** and 255.255.255.0 for the **Net mask**. Leave the checkboxes **BOOTP** and **DHCP** blank since neither a BOOTP server nor a DHCP server is used. It is furthermore not necessary to specify a **Gateway IP** address since the network does not have any connection to the outside world. Thus, the parameter settings for the first controller are as follows:

The screenshot shows a 'Device Parameters' dialog box with three tabs: 'IP Address', 'Ethernet', and 'Open Modbus'. The 'Ethernet' tab is selected. The 'Description' field contains the text '07 KT 98 Ethernet'. Below this, there are three checkboxes: 'DHCP' (unchecked), 'BOOTP' (unchecked), and 'IP address' (checked). Under the 'IP address' checkbox, there are three input fields: 'IP address' with the value '10 . 49 . 91 . 251', 'Net mask' with the value '255 . 255 . 255 . 0', and 'Gateway' with the value '0 . 0 . 0 . 0'. At the bottom of the dialog, there are three buttons: 'OK', 'Abbrechen', and 'Übernehmen'.

Switch to the **Ethernet** parameters group. For the interface select **Twisted Pair**. The structure of the network considered in this example is suitable for the use of the **Auto negotiation** function. All subscribers are connected to each other via switches and support the auto negotiation functionality. Therefore it is suitable to activate the auto negotiation function by checking the corresponding checkbox. Then all devices automatically agree during operation on the transmission parameters providing optimum performance (here: 100 Mbit/s full-duplex). If it

is desired to have a slower transmission rate or another duplex mode, the corresponding settings have to be done manually for all devices. For this example, however, the auto negotiation function is used for all devices. Thus, the Ethernet parameter settings are as follows:



The **OpenModbus** functionality is not used in this example. Therefore it is not required to set the corresponding parameters.

The application-specific adjustment of the device parameters is now completed. Continue as described under 4.2 "General procedure for configuring the coupler" and download the configuration data to the Ethernet coupler.

Repeat the creation and the download of the configuration data for the other two controllers accordingly. The individual configurations only differ in the set IP address.

4.5.3 Implementation in the user program

Configuring the couplers for fast data communication via UDP/IP is completed. Now the blocks for processing the protocol have to be implemented into the different user programs.

For this purpose, start the programming software 907 AC 1131 and create a new project for a controller 07 KT 98 or open a corresponding existing project. First select **Window | Library Manager | Additional Library** to integrate the files **Coupler_S90_V50.lib** (or higher version) and **Ethernet_S90_V50.lib** (or higher version) into the project. Then insert an instance of the ETH_AINIT block into your project.

Protocol processing initialization:

The procedure for initializing the protocol processing is identical for all controllers used in this example.

The **ETH_AINIT** block serves for the initialization of the protocol processing and the required resources. Furthermore the block outputs provide various information about the current status of the protocol processing. Protocol processing is only active as long as TRUE is applied at input EN of the ETH_AINIT block. Since in our example the protocol shall be permanently active, please immediately assign TRUE to input **EN**. The FALSE/TRUE edge required for initialization is automatically generated during the first program cycle. With this edge all other input values of the block are read. Later changes of the input values applied at the block inputs are ignored.



Warning:

The protocol for fast data communication via UDP/IP is only active while the user program is running. If the controller is stopped, protocol processing is finished automatically and restarted again via the ETH_AINIT block on the next start.

For controllers with only one Ethernet coupler the coupler is located in the first slot. Therefore assign the value 1 to block input **CONO**.

At input **LRH** the desired receive buffer length is specified in bytes. All received telegrams are first stored in this buffer and can then be read by means of the ETH_AREC block. Aside from the actual user data of a telegram the sender's IP address and the telegram length are stored. This results in 6 bytes overhead per received telegram. In this example it shall be possible to store up to 20 telegrams. Thus, input LRH must have a value of 280 $((6 + 8) \times 20 = 280)$ for the predetermined fixed user data length of 8 bytes.

At the inputs **LSHB** and **LSLB** the transmit buffer sizes for telegrams with high (LSHB) or low (LSLB) priority are specified in bytes. UDP telegrams can be transmitted either with normal or high priority. For each priority a separate transmit buffer is available. If only one transmit priority is used in the program, the corresponding transmit buffer has to be configured with a sufficient size. In this case the length of the respective other transmit buffer can be specified as 0. If telegrams with normal priority and telegrams with high priority are transmitted in the user program, both buffers have to be configured with sufficient size.

If mixed priorities are used, the telegrams with high priority are transmitted first. Telegrams with normal priority (if available) are not transmitted until the transmit buffer for telegrams with high priority does no longer contain any telegrams. If another message with high priority is written into the transmit buffer before the transmission of all telegrams with normal priority is finished, the normal priority transmission is interrupted for the transmission of the high priority message. After this the transmission of telegrams with normal priority is continued.

Aside from the actual user data, each telegram to be transmitted seizes in one of the transmit buffers 11 additional bytes for the telegram header. For this example only telegrams with normal priority shall be transmitted. The corresponding transmit buffer shall be able to store up to 12 telegrams temporarily. For the predetermined fixed user data length of 8 bytes the resulting values are 228 $((11 + 8) \times 12 = 228)$ for LSLB and 0 for LSHB.

At input **LSTOB** the desired timeout buffer length is specified in bytes. This buffer is used to automatically store information about telegrams which could not be transmitted to the recipient due to an error. Errors can be caused e.g. by an interrupted Ethernet line, an incorrectly specified IP address for the recipient which does not exist in the network or a recipient controller which is in STOP state. By default only the IP address of the recipient is stored here. If it is desired for better identification to additionally store parts of the user data of the failed transmit

process, the corresponding length has to be specified at input **LHEAD**. The timeout buffer can be read using the **ETH_ASTO** block.

In this example the first user data byte of the telegram which could not be delivered shall be stored in addition to the IP address of the recipient. For this purpose, the value 1 has to be specified for input **LHEAD**. Furthermore it shall be possible to store up to 6 data records. Thus, the resulting value for input **LSTOB** is 30 $((4 + 1) \times 6 = 30)$.

Input **ENBC** is used to specify whether received broadcast telegrams shall be stored in the receive buffer (**ENBC = TRUE**) or dismissed (**ENBC = FALSE**). In our example no broadcasts are used. Therefore, **FALSE** can be assigned to **ENBC**.

Block input **ENRA** is used to specify what should happen if further telegrams are received after the receive buffer is full. If **ENRA = FALSE** the oldest telegrams stored in the buffer are replaced by the received new telegrams. If **ENRA = TRUE** the data stored in the buffer are kept and the subsequent telegrams are dismissed instead until sufficient space is available again in the buffer.

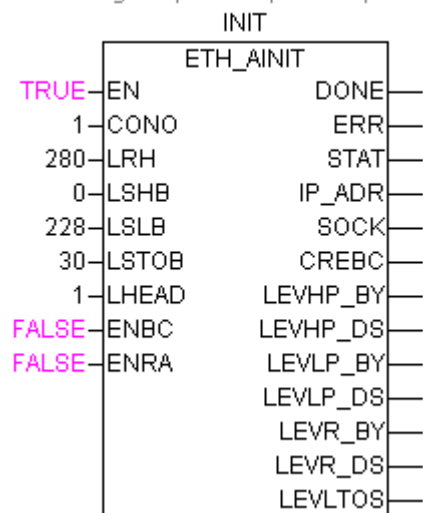


Note:

In order to avoid receive buffer overflow e.g. due to higher data traffic than expected, the receive buffer has to be enlarged or read more frequently via the **ETH_AREC** block.

Parameter assignment for the **ETH_AINIT** block is now completed. Represented in FBD it should look as follows:

Steuerungen 1, 2 und 3, PLCs 1, 2 and 3



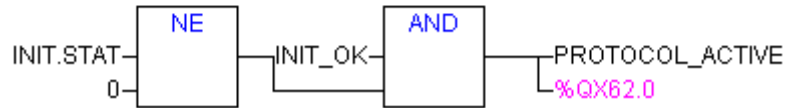
The block outputs provide various information about the protocol processing status. If **DONE = TRUE** and **ERR = 0**, the protocol could be initialized successfully. Furthermore, protocol processing is active as long as **STAT <> 0**. This information can be used to generally release further protocol handling.

For testing purposes the protocol processing status (active / inactive) is also displayed at output A 62,00 (%QX62.0) of the controller.

Steuerungen 1, 2 und 3, PLCs 1, 2 and 3



Steuerungen 1, 2 und 3, PLCs 1, 2 and 3

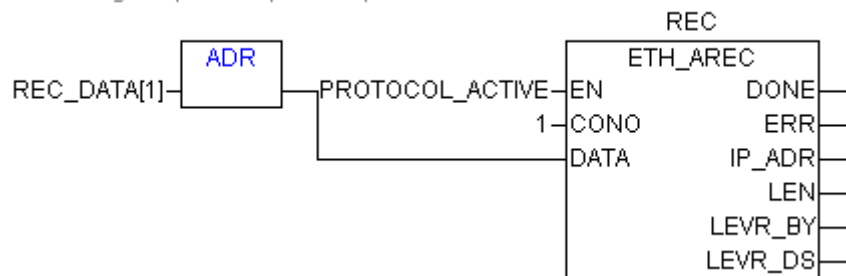


Reception of data:

Data reception is very similar for all controllers used in this example. The figures show a higher program part applicable for all controllers. In the programs for the individual controllers only a part of these evaluations has to be implemented.

Reading telegrams from the receive buffer is done using the **ETH_AREC** block. If necessary, it is also possible to use several instances of the **ETH_AREC** block within one program to enable faster reading of the telegrams. The block is processed if input **EN** = TRUE. If protocol processing is active, the receive block should also be active permanently. Therefore, the variable **PROTOCOL_ACTIVE** produced above has to be connected to input **EN**. According to the **ETH_AINIT** block the value 1 has to be specified at input **CONO**. The variable to be used to store a receive telegram has to be applied to block input **DATA** via the **ADR** operator. Since in this example all telegrams have a length of 8 bytes, a variable named **REC_DATA : ARRAY [1..8] OF BYTE** is created and its first element **REC_DATA[1]** is applied to the **ADR** operator. Thus, the following assignments are obtained for receive block **ETH_AREC**:

Steuerungen 1, 2 und 3, PLCs 1, 2 and 3

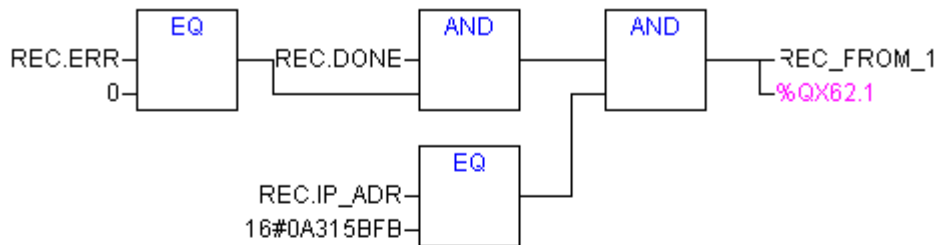


The values **TRUE** at output **DONE** and **0** at output **ERR** indicate that a telegram has been successfully read from the buffer during the present cycle and copied to the area specified at input **DATA**. The user data length of the telegram is output at **LEN**. The block always reads the next receive telegram stored in the buffer without taking into account the IP address of its sender. The sender's IP address of the respective telegram is output at **IP_ADR**. If, as shown in this example, telegrams are received from different controllers and if the data received from the respective controller shall not be overwritten by the data received from the other controllers, it is furthermore required to copy the data to an area depending of the sender's IP address. The distinction whether and from which controller a telegram has been received is in this example done by means of the connections shown below. Each byte in **IP_ADR** represents one octet of the IP address. Consequently 16#0A315BFC is the hexadecimal coded IP address

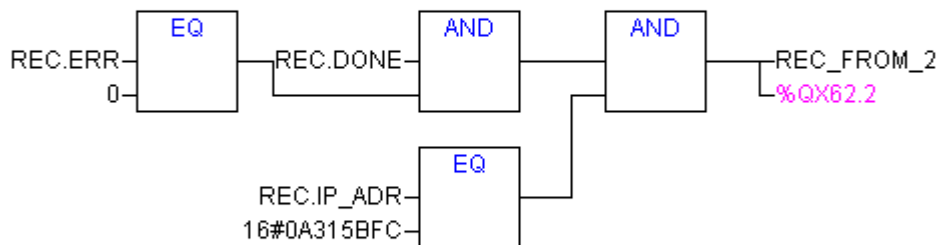
10.49.91.252 of the second controller and 16#0A315BFD is the IP address 10.49.91.253 of the third controller.

The figure below shows a higher program part for all controllers. In the programs for the individual controllers only a part of these evaluations has to be implemented. So, only REC_FROM_2 and REC_FROM_3 have to be evaluated in controller 1, only REC_FROM_1 and REC_FROM_3 in controller 2 and only REC_FROM_1 and REC_FROM_2 in controller 3. For testing purposes the reception of a telegram from a particular controller is indicated at a corresponding controller output.

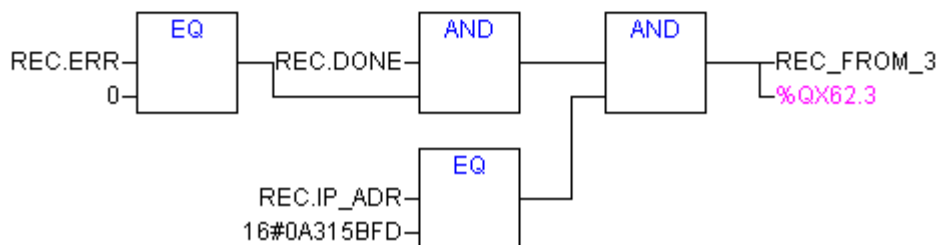
Steuerungen 2 und 3, PLCs 2 and 3



Steuerungen 1 und 3, PLCs 1 and 3



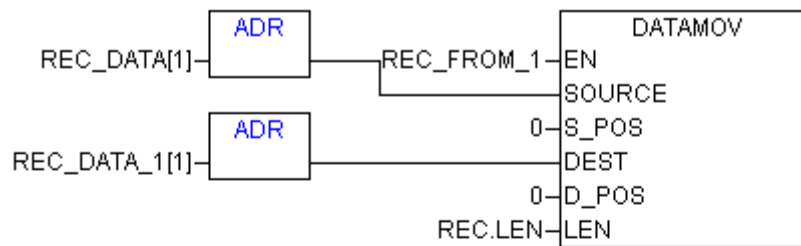
Steuerungen 1 und 2, PLCs 1 and 2



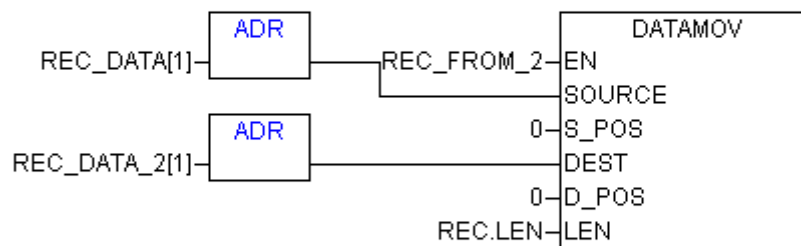
Using this receive detection it is now possible to copy the received data to the corresponding areas REC_DATA_1, REC_DATA_2 and REC_DATA_3. These variables are all of the type ARRAY [1..8] OF BYTE. For the purpose of copying the received user data you can use the **DATAMOV** function contained in the library **Com_S90_VXX**, for example.

Here it is also applicable that only two of the three networks represented below have to be implemented for the individual controllers depending on their controller number.

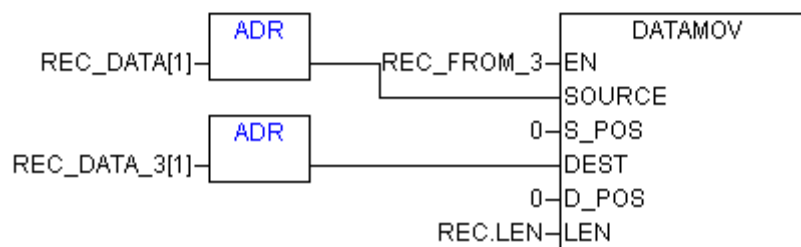
Steuerungen 2 und 3, PLCs 2 and 3



Steuerungen 1 und 3, PLCs 1 and 3



Steuerungen 1 und 2, PLCs 1 and 2

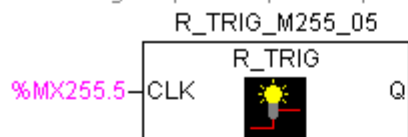


The most recent receive data of the respective controllers are now available for further processing in the corresponding ARRAYS.

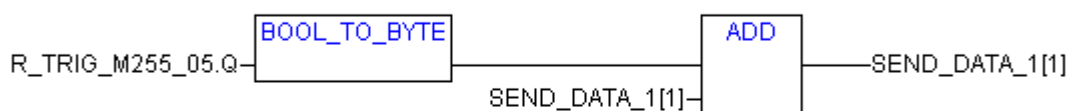
Sending of data:

For this example the user data contents of the transmit telegrams are simulated in all controllers. First define the variables containing the data to be transmitted. Since all telegrams shall contain 8 bytes of user data it is suitable to define the SEND_DATA_X variables as variables of the type ARRAY [1..8] OF BYTE. According to the number of the particular controller to which the data shall be sent, the variables SEND_DATA_2 and SEND_DATA_3 have to be specified in the program for controller 1, the variables SEND_DATA_1 and SEND_DATA_3 in the program for controller 2 and the variables SEND_DATA_1 and SEND_DATA_2 in the program for controller 3. In this example the first user data byte is incremented with each rising edge of the oscillator M 255,05 (%MX255.5).

Steuerungen 1, 2 und 3, PLCs 1, 2 and 3



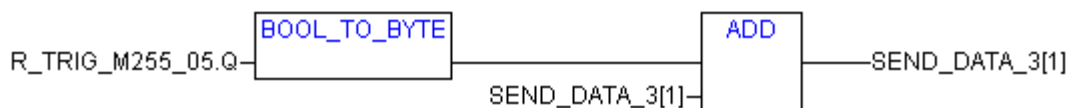
Steuerungen 2 und 3, PLCs 2 and 3



Steuerungen 1 und 3, PLCs 1 and 3



Steuerungen 1 und 2, PLCs 1 and 2

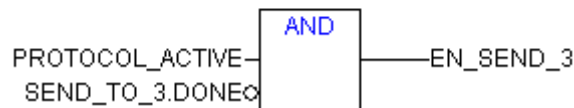
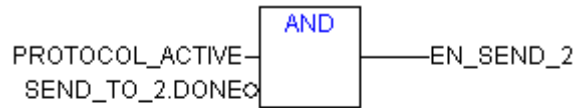


In contrast to the program parts for initializing the protocol processing and for processing the receive telegrams mentioned earlier which were nearly identical for all three controllers, the control for sending telegrams is considerably different for each individual controller. Therefore this program part is viewed separately for each controller.

The transmit process or the process of storing a telegram to be transmitted in the corresponding buffer is initiated by a rising edge applied at input **EN** of the **ETH_ASEND** block. The block confirms this process with a rising edge at output **DONE**. If an error occurred during this process (e.g. receive buffer full), this is indicated at output **ERR**. A telegram to be transmitted was only stored successfully in the corresponding transmit buffer if **DONE = TRUE** and **ERR = 0**.

Controller 1:

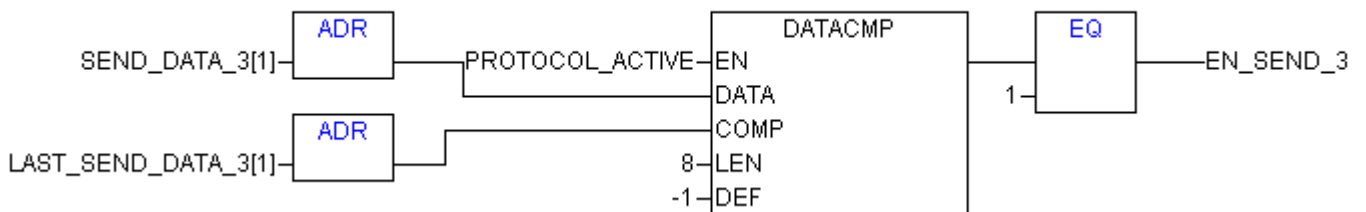
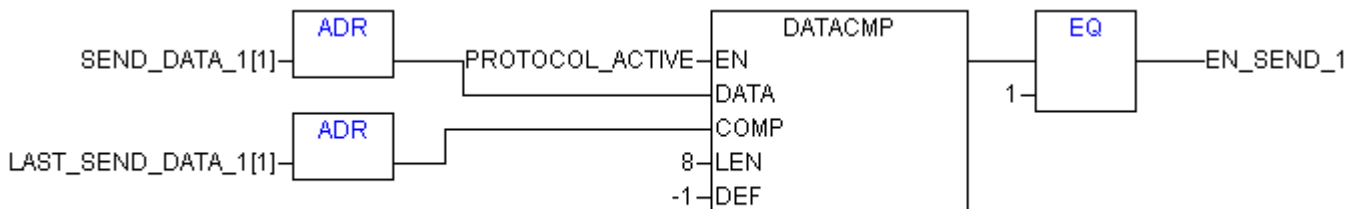
Controller 1 shall send its data periodically to each of the other two controllers. For this, it shall not be taken into account whether an error occurred when storing the telegram in the buffer. However, if an error occurs, the data are updated if necessary and transmitted the next time. For this, output DONE of the ETH_ASEND block can be used in conjunction with the general release to produce a FALSE/TRUE edge at input EN of the same block in order to initiate the transmission. Thus, the transmit processes in controller 1 are controlled as follows:



After this program part only the actual transmit blocks have to be inserted as described below.

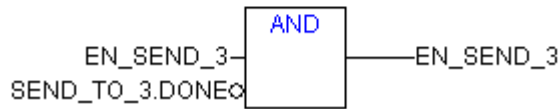
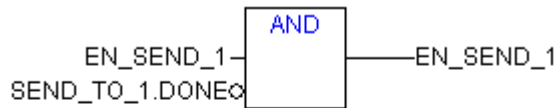
Controller 2:

Controller 2 shall only transmit its data to the other controllers if the data have changed compared with the data last transmitted to the respective controller. This shall avoid unnecessary load on the Ethernet network. Aside from the variable for the actual transmit data SEND_DATA_X, another data area LAST_SEND_DATA_X of the type ARRAY [1..8] OF BYTE has to be specified for each target controller to enable a comparison of the last transmitted data with the current transmit data. All ARRAYS are automatically initialized with 0 upon the program start. The data to be transmitted are always stored in SEND_DATA_X of the respective target controller. The contents comparison of the last transmitted data with the current data to be transmitted is done using the DATACMP function contained in the library Com_S90_Vxx.

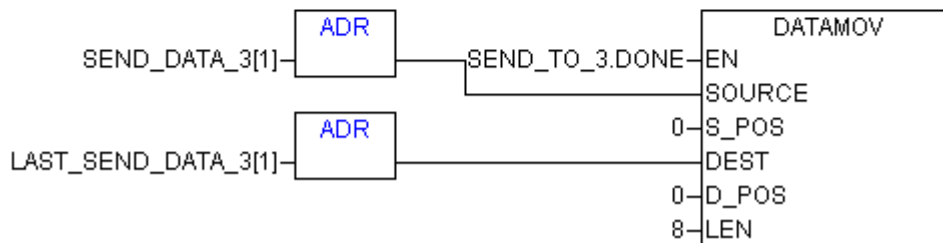
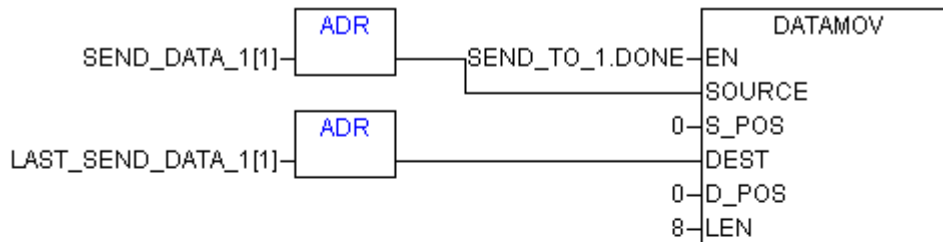


Storing the data in the transmit buffer is initiated by a rising edge at input EN of the ETH_ASEND block. If the data to be transmitted change between two subsequent program cycles, the comparison mentioned above causes the corresponding release EN_SEND_X to stay TRUE and no edge required for transmitting the data is produced. The required edge can

be forced by arranging that the releases are only set to TRUE if the corresponding data package has not been stored in the transmit buffer during the previous program cycle.



After this program part the actual transmit blocks have to be inserted as described below. In case of controller 2 an additional program part is required after activating the transmit blocks. Data shall only be transmitted to one of the other controllers if the data have changed since the last transmission. Consequently, after a data package has been stored in the transmit buffer successfully, the data additionally have to be taken over for the next transmit process for the purpose of comparison. For reasons of simplification no evaluation of output ERR of the ETH_ASEND block is performed here. Copying the data is done using the DATAMOV function contained in the library Com_S90_VXX.



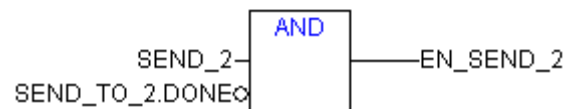
Controller 3:

Controller 3 shall only transmit its data to one of the other two controllers if it has received data from this controller before. For this purpose, the variables REC_FROM_X have already been specified for data reception where X is the number of the sending controller. These variables specify whether and by which controller a receive telegram was read from the buffer during this program cycle. However, these variables cannot be directly connected to input EN of the ETH_ASEND block since storing the data in the transmit buffer has to be initiated by a rising edge at input EN of the ETH_ASEND block. If a telegram is received from the same controller in two subsequent program cycles, the respective variable REC_FROM_X is in both cycles TRUE. The required edge can be forced by arranging that the releases are only set to TRUE if the corresponding data package has not been stored in the transmit buffer during the previous program cycle. In order to avoid the transmit request caused by the reception during the previous cycle from being lost, an additional variable (here: SEND_X) has to be set.

REC_FROM_1—[S]SEND_1



REC_FROM_2—[S]SEND_2



After this program part the actual transmit blocks have to be inserted as described below. In case of controller 3 an additional program part is required after activating the transmit blocks. The variable SEND_X set during data reception has to be reset again if a data package has been stored in the transmit buffer successfully.

SEND_TO_1.DONE—RSEND_1

SEND_TO_2.DONE—RSEND_2

The implementation of the transmit process control is now completed for all three controllers. Now it is only left to insert the actual ETH_ASEND transmit blocks into the programs.

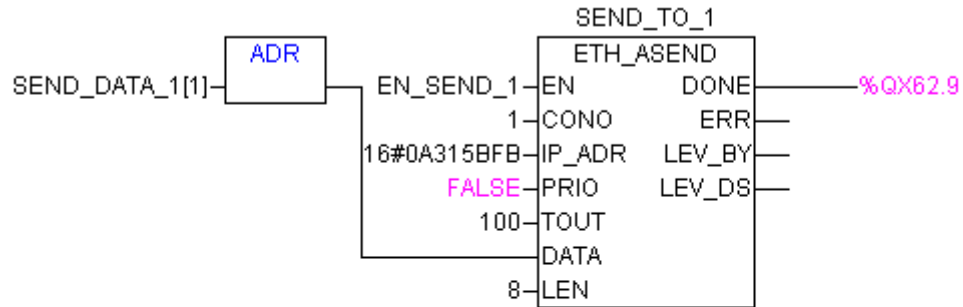
Transmission:

The transmit process or the process of storing a telegram to be transmitted in the corresponding buffer is initiated by a rising edge applied at input **EN** of the **ETH_ASEND** block. The block confirms this process with a rising edge at output **DONE**. If an error occurred during this process (e.g. receive buffer full), this is indicated at output **ERR**. A telegram to be transmitted was only stored successfully in the corresponding transmit buffer if **DONE = TRUE** and **ERR = 0**.

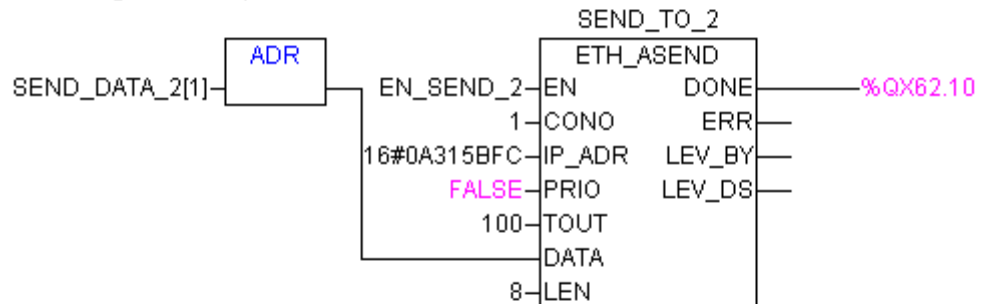
The initiation of the transmit processes is in all three controllers done by means of the variable EN_SEND_X where X is the number of the controller to which the transmission shall be directed. Controller 1 is sending to controller 2 (EN_SEND_2) and 3 (EN_SEND_3), controller 2 is sending to controller 1 (EN_SEND_1) and 3 (EN_SEND_3) and controller 3 is sending to controller 1 (EN_SEND_1) and 2 (EN_SEND_2). The data to be transmitted were stored in the byte ARRAYs SEND_DATA_X where X is likewise the number of the target controller. Furthermore, normal priority, a timeout value of 100 ms and a user data length of 8 bytes were predetermined for all telegrams. Thus, the following assignments are obtained for the transmit blocks. For testing purposes the transmission of a telegram to a particular target controller is additionally indicated at a corresponding output of the respective sending controller.

According to the number of the particular controller to which the data shall be sent, only the variables SEND_TO_2 and SEND_TO_3 have to be specified in the program for controller 1, the variables SEND_TO_1 and SEND_TO_3 in the program for controller 2 and the variables SEND_TO_1 and SEND_TO_2 in the program for controller 3.

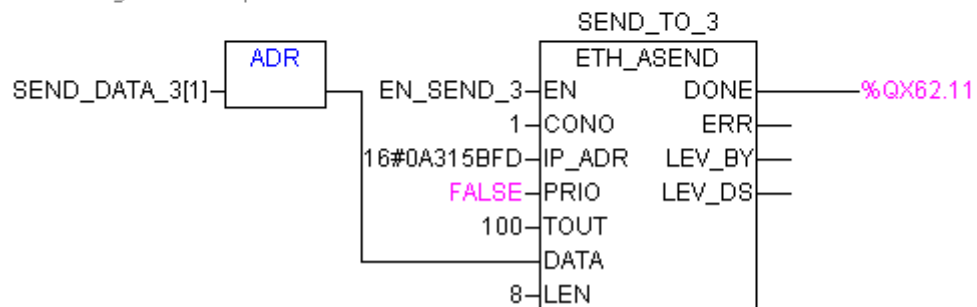
Steuerungen 2 und 3, PLCs 2 and 3



Steuerungen 1 und 3, PLCs 1 and 3



Steuerungen 1 und 2, PLCs 1 and 2

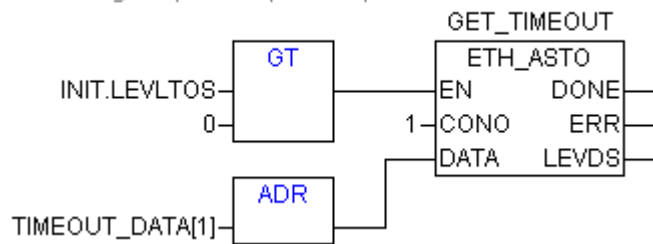


If a telegram stored in the transmit buffer could not be transmitted to the target controller for any reason or if the target controller did not acknowledge the reception of the telegram within the specified send timeout of 100 ms, the information about this telegram specified during the initialization are stored in the timeout buffer. In case of a buffer overflow the respective oldest entry is overwritten and the number of entries still stays at the maximum value (here: 6). The current number of failed transmit telegrams stored in this buffer can be evaluated via output **LEVLTO5** of the **ETH_AINIT** block.

If the individual controllers are started one after the other it can happen that the value of LEVLTO5 first increases in the controllers already started since the controllers which are not yet in RUN state are not yet able to process the protocol. Then, after all controllers are in RUN state and if the Ethernet network is connected correctly and not overloaded the LEVLTO5 values should no longer change in any controller.

If necessary, the information about the undeliverable transmit telegrams stored in the timeout buffer can be read via the **ETH_ASTO** block and evaluated. The figure below shows the corresponding block call for all controllers.

Steuerungen 1, 2 und 3, PLCs 1, 2 and 3



If output **DONE** is TRUE and **ERR** is 0, a data record has been read from the buffer and stored in TIMEOUT_DATA. If necessary, it is now possible to further evaluate the data record. Furthermore, the number of entries in this buffer is decremented if no other timeout occurred in the meantime. According to the initialization of the protocol processing TIMEOUT_DATA[1] to [4] contain the IP address of the target controller and TIMEOUT_DATA[5] contains the first byte of the original user data.

Download the projects to the corresponding controllers. The implementation of fast data communication via UDP/IP is now completed.

5 Used terms and explanations

5.1 Used terms

- ACR Attenuation to Crosstalk Ratio
- ARP Address Resolution Protocol
- AUI Attachment Unit Interface
- BOOTP Bootstrap Protocol
- CSMA/CD Carrier Sense Multiple Access / Collision Detection
- DHCP Dynamic Host Configuration Protocol
- DNS Domain Name Service
- DoD Department of Defense
- ELFEXT Equal Level Far End Cross Talk
- FEXT Far End Cross Talk
- FTP File Transfer Protocol
- http Hypertext Transfer Protocol
- ICMP Internet Control Message Protocol
- IEEE Institute of Electrical and Electronical Engineers
- IFG Interframe Gap
- IP Internet Protocol
- LAN Local Area Network
- LLC Logical Link Control
- MAC Media Access Control Protocol
- MDI Medium Dependent Interface
- MDI-X Medium-Dependent Interface, crossed
- NEXT Near End Cross Talk
- NIS Network Information Service
- PLS Physical Layer Signaling
- PMA Physical Medium Attachment
- PPP Point-to-Point-Protocol

- SMTP Simple Mail Transfer Protocol
- STP Shielded Twisted Pair
- TCP Transmission Control Protocol
- TOS Type of Service
- TP Twisted Pair
- TTL Time to Live
- UDP User Datagram Protocol
- URL Uniform Resource Locator
- UTP Unshielded Twisted Pair
- WAN Wide Area Network
- WWW World Wide Web

5.2 Explanations

- IP is connectionless
- UDP is connectionless
- TCP is connection oriented (secure communication)
- IP implements the transmission of normal datagrams
- ICMP serves for the transmission of error and information messages
- An IP module of a device can have several IP addresses
- Sockets consist of an IP address and a port and are the interface between TCP and the above application
- Ports are (partly standardized) symbolic numbers
- Transmission of data by an application is performed via an actively open socket
- Reception of data by an application in the target computer is performed by assigning the receive telegram
- (IP address/port) to a passively open socket
- TCP communication phases:
Connection setup -> data transmission -> closing of connection
- FTP:
File Transfer Protocol, protocol used for the transfer of files
- HTTP:
Hyper Text Transfer Protocol, protocol used for the transfer of Internet pages (webservers)
- SMTP:
Protocol used for sending Emails
- POP3:
Protocol used for the reception of Emails via an Email server
- Hub:
Simple "distribution box" for star topology, received telegrams are forwarded to all connected subnetworks
- Switch:
Intelligent "distribution box" for star topology with knowledge about the structure and the subscribers of the connected subnetworks, specific forwarding of received telegrams to the concerning connected subnetwork or subscriber which reduces the load within the network compared with hubs
- Router, server:
Connection of a local network (LAN, e.g. Ethernet) to the "outside world" (WAN, e.g. DSL)
- DHCP:
Protocol for the automatic IP address assignment performed by the server

- BOOTP:
Protocol for the automatic IP address assignment performed by the server

6 Figures

Figure 1-1: Structure of an Ethernet 802.3 frame	1-2
Figure 1-2: Ethernet in the ISO/OSI model	1-4
Figure 1-3: Ethernet protocols	1-5
Figure 1-4: Principle structure of an IP header	1-7
Figure 1-5: IP datagram in the Ethernet frame	1-7
Figure 1-6: Course of the transfer processes	1-14
Figure 1-7: Principle of the logical connections between pairs of sockets	1-15
Figure 1-8: Principle structure of a TCP header	1-16
Figure 1-9: TCP segment in the Ethernet frame	1-16
Figure 1-10: Simplified state diagram of a TCP module	1-17
Figure 1-11: Principle structure of a UDP header	1-20
Figure 1-12: Comparison between UDP and TCP data packages	1-20
Figure 1-13: Structure of an OpenModbus on TCP/IP telegram	1-21
Figure 1-14: Structure of the modbus application protocol header	1-22
Figure 1-15: FTP command syntax	1-29
Figure 1-16: Top Level Domains of the USA	1-30
Figure 1-17: Pin assignment of RJ45 sockets	1-33
Figure 1-18: Wiring of a crossover cable	1-34
Figure 1-19: Wiring of a 1:1 cable	1-34
Figure 1-20: Example of a simple network	1-35
Figure 1-21: Stackable hubs	1-37
Figure 1-22: Use of a class II hub	1-38
Figure 1-23: Use of hubs and switches	1-40
Figure 1-24: Redundant network structure using switches	1-41
Figure 3-1 : Hierarchy model	3-2
Figure 3-2 : Redundant model	3-3
Figure 3-3 : Safe models	3-4
Figure 3-4 : Direct connection of all stations to switches	3-6
Figure 4-1 : Example for OpenModbus on TCP/IP - controller operated as server	4-8
Figure 4-2 : Example for OpenModbus on TCP/IP - controller operated as client	4-12
Figure 4-3 : Example configuration for data communication via UDP/IP	4-19

7 Tables

Table 1-1: Classes of IP addresses..... 1-8

Table 1-2: Overview of IP address structures..... 1-9

Table 1-3: Well-known ports..... 1-15

Table 1-4: Fixed port numbers (assigned numbers) for standard processes 1-19

Table 1-5: Message types of the DHCP protocol 1-25

Table 1-6: Specified cable properties 1-32

Table 1-7: Color coding of TP cables..... 1-32

Table 1-8: Pin assignment of RJ45 connectors..... 1-33

Table 2-1: Pin assignment of the attachment plug for the Ethernet cable..... 2-3

Table 2-2: Status LEDs..... 2-5

Table 3-1 : Data rate at 10 Mbit/s 3-6

Table 3-2 : Data rate at 100 Mbit/s 3-6

Table 3-3 : Net bandwidth at 100 Mbit/s 3-7

Table 3-4: Efficiency of data transmission 3-7

A

Address Resolution Protocol (ARP)
dynamic address mapping 1-27
static address mapping 1-27

B

BootP and DHCP
automatic IP address assignment 1-24
Bootstrap Protocol (BootP) 1-23
Dynamic Host Configuration Protocol (DHCP) 1-24
dynamic IP address assignment 1-24
function 1-25
manual IP address assignment 1-24

E

Ethernet S90
1:1 cables and crossover cables 1-34
10 Mbit/s hubs 1-37
10/100 Mbit/s dual-speed hubs 1-38
100 Mbit/s hubs 1-38
Address Resolution Protocol (ARP) 1-27
auto negotiation 1-3
BootP and DHCP 1-23
bridges, switches and switching hubs 1-39
bus access method 1-3
cable length restrictions 1-34
CIF TCO/IP driver 4-4
concepts for structuring a network 3-1
configuring the coupler - general procedure 4-1
configuring the coupler for Modbus on TCP/IP (client)
4-13
configuring the coupler for Modbus on TCP/IP (server)
4-9
configuring the coupler for online access 4-5
Configuring the Ethernet couplers for UDP/IP 4-19
connection and transfer media 2-3
connector pin assignment 1-33
design examples 4-1
designing and planning a network 3-1
diagnosis 2-5
Domain Name Service Protocol (DNS) 1-30
Ethernet and TCP/IP 1-4
Ethernet used for online access 4-5
example configuration for UDP/IP 4-18
explanations 5-3
fast data communication via UDP/IP 4-18
features 2-1
File Transfer Protocol (FTP) 1-28
frame formats 1-2
gateway driver 4-3
gateways 1-42
half duplex and full duplex 1-3

hierarchy model 3-2
hosts file 1-31
Hypertext Transfer Protocol (HTTP) 1-28
implementation 2-4
Internet Control Message Protocol (ICMP) 1-12
Internet Name Service (EIN 116) 1-30
Internet Protocol (IP) 1-6
MAC address 1-2
media converters 1-41
Modbus on TCP/IP - example 4-8
network cables 1-32
network components 1-35
Network Information Service (NIS) 1-30
OpenModbus on TCP/IP 1-21
operation as Modbus on TCP/IP client 4-12
operation as server / slave 4-8
Point-to-Point Protocol (PPP) 1-5
protocols and applications 1-5
redundant model 3-3
repeaters and hubs 1-36
restrictions 2-2
routers 1-42
safe models 3-4
Simple Mail Transfer Protocol (SMTP) 1-29
sockets 2-2
supported protocols 2-1
technical data 2-2
terminal devices 1-35
the Ethernet coupler 2-1
Transmission Control Protocol (TCP) 1-14
used terms 5-1
used terms and explanations 5-1
User Datagram Protocol (UDP) 1-19
user program for Modbus on TCP/IP (server) 4-12
user program for UDP/IP 4-20
user program Modbus on TCP/IP (client) 4-16
utilization and performance 3-5

I

Internet Control Message Protocol (ICMP)
address format request/reply message 1-13
destination unreachable message 1-13
echo request/reply message 1-12
information request/reply message 1-12
parameter problem message 1-14
redirect message 1-13
router discovery message 1-13
source quench message 1-13
time exceeded message 1-13
timestamp request/reply message 1-12
Internet Protocol (IP)
gateway 1-12
IP address assignment 1-9
IP addresses 1-8
IP header 1-7
special IP address 1-10

subnet mask 1-10

O

OpenModbus on TCP/IP
telegram structure 1-21

T

Transmission Control Protocol (TCP)

close timeout 1-19
connect timeout 1-18
give-up timeout / send timeout 1-18
inactive timeout 1-18
keep alive timeout 1-18
phases of a communication 1-17
port 1-14
retransmission timeout 1-18
socket 1-15
TCP header 1-16

U

User Datagram Protocol (UDP)

port 1-19
socket 1-20
UDP header 1-20

Content

1	Error messages of the internal couplers	K 1-1
1.1	Range of validity	1-1
1.2	Structure of the error messages of the internal couplers	1-1
1.2.1	Error identifier	1-1
1.2.2	Detailed info 1 – Card number	1-1
1.2.3	Detailed info 2 – Error class	1-2
1.2.4	Detailed info 3 – Cause of errors	1-3
1.2.5	Detailed infos 4 / 5 / 6 - Additional information	1-6
1.3	FK2 - Serious errors	1-10
1.4	FK3 - Light errors	1-11
1.5	FK4 Warnings	1-13
2	Tables	2-1
3	Index	I

1 Error messages of the internal couplers

1.1 Range of validity

The described error messages are valid for all internal couplers, **except ARCNET**.

1.2 Structure of the error messages of the internal couplers

Error class	Error identifier	Detailed info 1	Detailed info 2	Detailed info 3	Detailed info 4	Detailed info 5	Detailed info 6	Detailed info 7
FK2 / FK3 / FK4	200 _D C8 _H (Error identifier of internal couplers)	ID (Card number with error)	Error class of internal couplers	Cause of error	Additional information	Additional information	Additional information	0

Table 1-1: Structure of the error messages of the internal couplers

1.2.1 Error identifier

All error messages, concerning the internal couplers, have the error identifier **200_D or C8_H**.

1.2.2 Detailed info 1 – Card number

With all errors, the **detailed info 1** contains the **card number (ID)** of the involved coupler

The only **exceptions** are errors, which occur when reserving memory or providing resources, which apply to all couplers or the appearance of an invalid ID, which is not supported by the target system, e.g. in case of configuration. In case of general memory errors the detailed info 1 contains the value 0, in case of an invalid card number (ID), these invalid value is displayed.

1.2.3 Detailed info 2 – Error class

Besides the classification into the global error classes FK1 to FK4, the errors of the internal couplers are classified deeper in the detailed info 2. The following error classes are defined:

Detailed info 2		Error class field bus couplers	
Dec	Hex		
256 _D	0100 _H	Initialization error	Error when initializing the coupler or the driver in the PLC
512 _D	0200 _H	Operating system error	Operating system of the coupler reports an error
768 _D	0300 _H	Task1 status	Task1 of the coupler has an improper status
1024 _D	0400 _H	Task2 status	Task2 of the coupler has an improper status
1280 _D	0500 _H	Task3 status	Task3 of the coupler has an improper status
1536 _D	0600 _H	Task4 status	Task4 of the coupler has an improper status
1792 _D	0700 _H	Task5 status	Task5 of the coupler has an improper status
2048 _D	0800 _H	Task6 status	Task6 of the coupler has an improper status
2304 _D	0900 _H	Task7 status	Task7 of the coupler has an improper status
2560 _D	0A00 _H	Run-time error	Error of the coupler or driver in the PLC during running operation
2816 _D	0B00 _H	Planning error	The project contains none or faulty configuration data
65535 _D	FFFF _H	Fatal error	

Table 1-2: Detailed info 2 – Error class

1.2.4 Detailed info 3 – Cause of errors

In the detailed information 3 the cause of the field bus coupler error is more precisely specified. The following table shows the generally applicable error causes. Operating system errors or task errors are excluded here. The codes of these errors are listed further down.

General

Detailed info 3		Cause of error
Dec	Hex	
1 _D	01 _H	DP RAM of the coupler not accessible
2 _D	02 _H	Error when testing the coupler watchdog
3 _D	03 _H	Invalid size of the coupler DP RAM memory
4 _D	04 _H	Coupler not found
5 _D	05 _H	Incorrect coupler type (different coupler type than installed on delivery)
6 _D	06 _H	Incorrect coupler model (different participant type and/or different protocol than installed on delivery)
7 _D	07 _H	Unknown coupler
8 _D	08 _H	Heap error (control has not enough memory to apply the required resources)
9 _D	09 _H	Coupler has no or invalid configuration data
10 _D	0A _H	Life identifier error of the coupler (the coupler did not handle the life identifier)
12 _D	0C _H	Reset error (coupler did not carry out the reset request within the predetermined time)
13 _D	0D _H	Ready error (coupler did not signalize readiness for operation within the predetermined time)
14 _D	0E _H	Message was deleted (message was not processed within the predetermined time)
16 _D	10 _H	Sending mailbox error (no buffer available)
17 _D	11 _H	Receiving mailbox error (coupler did not respond or did respond with an error to the request message)
18 _D	12 _H	The called coupler is not a DP master (the project contains wrong configuration data)
19 _D	13 _H	The called coupler is not a DP slave (the project contains wrong configuration data)
25 _D	19 _H	Invalid configuration data
26 _D	1A _H	Invalid card number (the project contains wrong configuration data)
29 _D	1D _H	Loaded configuration data will not be accepted. The coupler has already 907 AC 1131 configuration data
30 _D	1E _H	Coupler signals error (ERROR LED), detailed diagnosis via connection element STAT or configurator

Table 1-3: Detailed info 3 – Cause of errors, general

Operating system errors and task errors of the coupler

Detailed info 3		Cause of error
Dec	Hex	
1 _D	01 _H	Task is blocked
2 _D	02 _H	Task does not transfer process data
10 _D	0A _H	Initialization error
..	..	Initialization error
49 _D	31 _H	Initialization error
50 _D	32 _H	Faulty initialization base
51 _D	33 _H	Initialization error
99 _D	63 _H	Initialization error
100 _D	64 _H	UART parity error
101 _D	65 _H	UART framing error
102 _D	66 _H	UART overrun error
103 _D	67 _H	Incorrect number of data
104 _D	68 _H	Check sum error
105 _D	69 _H	Timeout
106 _D	6A _H	Protocol error
107 _D	6B _H	Data error
108 _D	6C _H	No acknowledgment
109 _D	6D _H	Initialization error
110 _D	6E _H	Incorrect protocol base
111 _D	6F _H	Initialization error
..	..	Initialization error
149 _D	95 _H	Initialization error
150 _D	96 _H	Error in message header
151 _D	97 _H	Incorrect message length
152 _D	98 _H	Incorrect message command
153 _D	99 _H	Incorrect message structure
154 _D	9A _H	Message error
155 _D	9B _H	Message timeout
156 _D	9C _H	Initialization error
157 _D	9D _H	Initialization error
158 _D	9E _H	Initialization error
159 _D	9F _H	Initialization error
160 _D	A0 _H	Incorrect telegram header
161 _D	A1 _H	Incorrect module address in message
162 _D	A2 _H	Incorrect data area identifier in message
163 _D	A3 _H	Incorrect data address in message
164 _D	A4 _H	Incorrect data index in message
165 _D	A5 _H	Incorrect number of data in message
166 _D	A6 _H	Incorrect data type in message
167 _D	A7 _H	Incorrect function in message
168 _D	A8 _H	Initialization error
169 _D	A9 _H	Initialization error
170 _D	AA _H	Error in message base

Table 1-4: Detailed info 3 – Cause of errors, Operating system errors and task errors of the coupler

Detailed info 3		Cause of error
Dec	Hex	
171 _D	AB _H	Initialization error
..	..	Initialization error
199 _D	C7 _H	Initialization error
200 _D	C8 _H	Task not initialized
201 _D	C9 _H	Task blocked
202 _D	CA _H	Segment error
203 _D	CB _H	User error
204 _D	CC _H	Initialization error
..	..	Initialization error
209 _D	D1 _H	Initialization error
210 _D	D2 _H	Data base error
211 _D	D3 _H	Data base write error
212 _D	D4 _H	Database read error
213 _D	D5 _H	Faulty structure
214 _D	D6 _H	Faulty parameter
215 _D	D7 _H	Faulty configuration
216 _D	D8 _H	Faulty function list
217 _D	D9 _H	System error
218 _D	DA _H	Initialization error
219 _D	DB _H	Initialization error
220 _D	DC _H	System data base error
221 _D	DD _H	Initialization error
..	..	Initialization error
239 _D	EF _H	Initialization error
250 _D	FA _H	Task in initialization state
251 _D	FB _H	Task is waiting for initialization
252 _D	FC _H	Task is activated
253 _D	FD _H	Task implemented
254 _D	FE _H	Task in configuration state
255 _D	FF _H	Task not found

Table 1-4 continued: Detailed info 3 – Cause of errors, Operating system errors and task errors of the coupler

1.2.5 Detailed infos 4 / 5 / 6 - Additional information

In the detailed infos 4, 5, and 6 further additional information is displayed, which allows a more exact diagnosis of the cause of error. The meaning and the contents of this information depends on the cause of error.

Types of couplers

Detailed infos 4 / 5		Type
Dec	Hex	
55 _D	37 _H	Coupler with permanent mode of operation
57 _D	39 _H	Coupler with variable mode of operation

Table 1-5: Detailed infos 4 / 5 - Types of couplers

Coupler models

Detailed infos 4 / 5		Model
Dec	Hex	
49 _D	31 _H	PROFIBUS master/slave
51 _D	33 _H	PROFIBUS DP master
52 _D	34 _H	Reserved
56 _D	38 _H	Interbus master
76 _D	4C _H	DeviceNet master

Table 1-6: Detailed infos 4 / 5 - Coupler models

Memory errors

Detailed info 4		Detailed info 5		Model
Dec	Hex	Dec	Hex	
00 _D	00 _H	00 _D	00 _H	Not enough memory for saving the 907 AC 1131 PROFIBUS configuration data
00 _D	00 _H	31 _D	1F _H	Not enough memory for buffering the SMC configuration data
00 _D	00 _H	33 _D	21 _H	Not enough memory for providing general driver resources
Card number		00 _D	00 _H	Not enough memory for providing the process data input descriptions
Card number		01 _D	01 _H	Not enough memory for providing the process data output descriptions
Card number		32 _D	20 _H	Not enough memory for providing the message mailboxes
Card number		33 _D	21 _H	Not enough memory for providing the resources

Table 1-7: Detailed infos 4 / 5 - Memory errors

Errors during loading the configuration data

Detailed info 4		Detailed info 5		Model
Dec	Hex	Dec	Hex	
00 _D	00 _H	00 _D	00 _H	Configuration data have invalid format
06 _D	06 _H	00 _D	00 _H	Configuration data does not fit to the coupler model
07 _D	07 _H	00 _D	00 _H	Configuration data are of unknown type

Table 1-8: Detailed infos 4 / 5 - Errors during loading the configuration data

Request and response identifiers in messages

Several services are processed message controlled. If an error occurs, it is displayed. The faulty message can be identified by means of the request or response identifier and the mode. If the request identifier does not equal 0 and the response identifier equals 0, it refers to a message, which is addressed from a connection element in the user program, the system or the configurator to the coupler or to a message, which is initiated by the coupler itself. If the response identifier does not equal 0 and the request identifier equals 0, it refers to a response of the coupler to a preceding request.

Detailed info 4		Detailed info 5		Meaning
Dec	Hex	Dec	Hex	
6 _D	00 _H	2 _D	02 _H	Reading the configuration data from the Flash of the coupler
6 _D	06 _H	3 _D	03 _H	Writing the configuration data to the Flash of the coupler
6 _D	06 _H	4 _D	04 _H	Deleting the configuration data in the Flash of the coupler
6 _D	06 _H	5 _D	05 _H	Reading the assignment of the Flash of the coupler
15 _D	0F _H	Slave address		Rereading the I/O data structure of a slave
67 _D	43 _H	Slave address		Initiation of the download of the 907 AC 1131 slave configuration data to a master
68 _D	44 _H	Slave address		Download of the 907 AC 1131 slave configuration data to a master
68 _D	44 _H	127 _D	7F _H	Download of the 907 AC 1131 bus parameters to a master
69 _D	45 _H	Slave address		Terminating of the download of the 907 AC 1131 slave configuration data to a master

Table 1-9: Detailed info 4/5 - Request and response identifiers in messages

Message errors

Detailed info 6		Error
Dec	Hex	
2 _D	02 _H	Counter-station does not provide for the required resources
3 _D	03 _H	Service not available
8 _D	08 _H	Service in counter-station not available
9 _D	09 _H	Response of the counter-station contains no data / invalid attribute in the counter-station
11 _D	0B _H	Command mode in the counter-station is already active
12 _D	0C _H	Conflict with the state of the object in the counter-station
14 _D	0E _H	Attribute in counter-station cannot be written
15 _D	0F _H	Counter-station refuses access
16 _D	10 _H	Service cannot be executed on the actual state of the counter-station
17 _D	11 _H	No response of the counter-station
18 _D	12 _H	Master not in the logical token ring
19 _D	13 _H	Coupler not in the offline configuration mode / insufficient number of data received
20 _D	14 _H	Attribute in counter-station not supported
21 _D	15 _H	Faulty parameters in request / data segment not configured / too many data received
22 _D	16 _H	Numbering error during loading the data / object does not exist in counter-station
23 _D	17 _H	Loaded data are longer than expected / overflow of the internal buffer in the counter-station
24 _D	18 _H	Sequence error during loading
25 _D	19 _H	Not plausible reaction of the communication partner / Check sum error during loading the data
26 _D	1A _H	Access protection during reading or writing the data segment
27 _D	1B _H	Segment not released for reading or writing
29 _D	1D _H	Coupler has no boot strap loader

Table 1-10: Message errors

Detailed info 6		Error
Dec	Hex	
40 _D	28 _H	No vacant command
41 _D	29 _H	Unknown command
42 _D	2A _H	Unknown command mode
43 _D	2B _H	Faulty command parameter
44 _D	2C _H	Message length and expected number of parameters are not identical
48 _D	30 _H	Timeout / counter station not configured, communication channel is not established
50 _D	32 _H	Coupler Flash is blocked / Faulty telegram format of the counter station
51 _D	33 _H	Error during deleting the Flash of the coupler
52 _D	34 _H	Unknown Area code / error during writing on the Flash of the coupler
53 _D	35 _H	Area exceeding / Flash of the coupler not configured
54 _D	36 _H	Access refused / timeout of the Flash of the coupler / service already in process
55 _D	37 _H	Faulty parameters in the data / access protection during deleting the Flash of the coupler / invalid address
57 _D	39 _H	Sequence error / two of same bus addresses detected
58 _D	3A _H	Requested data not available
59 _D	3B _H	Data not complete or faulty / multiple assignment of bus addresses
60 _D	3C _H	Faulty length indicator
61 _D	3D _H	Inadmissible predefined connection
62 _D	3E _H	Inadmissible predefined connection
63 _D	3F _H	Inadmissible predefined connection
64 _D	40 _H	Faulty length indicator
65 _D	41 _H	Faulty number of inputs
66 _D	42 _H	Faulty number of outputs
67 _D	43 _H	Unknown data type
68 _D	44 _H	Difference in data type description
69 _D	45 _H	Invalid memory address of the output data
70 _D	46 _H	Invalid memory address of the input data / slave is already configured
71 _D	47 _H	Unknown predefined connection / Faulty length of the data set
72 _D	48 _H	Inadmissible parallel connection / Faulty length of the configuration data
73 _D	49 _H	Inadmissible data rate / Faulty length of the expanded data
74 _D	4A _H	Faulty length of the PCP field
75 _D	4B _H	Data set inconsistent
76 _D	4C _H	Expanded data set inconsistent
77 _D	4D _H	Invalid memory address of the output data
78 _D	4E _H	Invalid memory address of the input data
79 _D	4F _H	Exceeding the maximal number of input and output data
80 _D	50 _H	Difference in the number of modules
81 _D	51 _H	Difference in the number of output modules
82 _D	52 _H	Difference in the number of input modules
83 _D	53 _H	Length difference in the output modules
84 _D	54 _H	Length difference in the input modules
85 _D	55 _H	Address conflict in the output data
86 _D	56 _H	Address conflict in the input data
87 _D	57 _H	Difference between ID code and input modules
88 _D	58 _H	Difference between ID code and output modules
89 _D	59 _H	Difference between ID code and output modules

Table 1-10 continued: Message errors

Detailed info 6		Error
Dec	Hex	
90 _D	5A _H	Difference between ID code and input modules
91 _D	5B _H	Invalid configuration level of the slave
92 _D	5C _H	Unknown length identifier
93 _D	5D _H	No data to slave available
94 _D	5E _H	Modification of the data set in this state not admissible
95 _D	5F _H	Difference in the data sets
115 _D	73 _H	Status data exceed the capacity of the diagnosis buffer
116 _D	74 _H	No diagnosis buffer available
119 _D	77 _H	DPV1 class 1 services deactivated
120 _D	78 _H	Maximal permissible number of alarms exceeded
121 _D	79 _H	Alarm deactivated
123 _D	7B _H	Inadmissible length of the alarm data
125 _D	7D _H	Inadmissible sequence number
126 _D	7E _H	DPV1 deactivated
127 _D	7F _H	Inadmissible format of the status PDU
128 _D	80 _H	Inadmissible length of the status PDU
129 _D	81 _H	DPV1 communication is stopped
130 _D	82 _H	Unit related diagnosis in the actual mode inadmissible / DPV1 communication aborted automatically
131 _D	83 _H	Inadmissible length of the unit related diagnosis / service not yet terminated
132 _D	84 _H	Inadmissible length of the module related diagnosis / inadmissible length indicator
133 _D	85 _H	Channel related diagnosis has unknown reference / invalid command parameter
134 _D	86 _H	Diagnosis data contain several revision numbers / alarm handling not initiated
135 _D	87 _H	No parameter data available / alarm handling stopped
136 _D	88 _H	Alarm to be acknowledged is not activated in the slave
137 _D	89 _H	Alarm to be acknowledged is not active
138 _D	8A _H	DPV1 class 2 services deactivated
139 _D	8B _H	DPV1 class 2 services already activated
140 _D	8C _H	DPV1 class 2 services actually deactivated
141 _D	8D _H	Invalid data in the RM_Registry
142 _D	8E _H	Invalid SAP in the Res_SAP_List
143 _D	8F _H	Invalid number of entries in the RM_Registry
144 _D	90 _H	Invalid number of entries in the Res_SAP_List
145 _D	91 _H	Length difference in DPV1 class 2 command
146 _D	92 _H	FDL error
147 _D	93 _H	Unknown or not connected SAP
148 _D	94 _H	No configuration data available
152 _D	98 _H	Unknown command
154 _D	9A _H	Unknown command
161 _D	A1 _H	Out of the valid range
165 _D	A5 _H	Length differences within the message
167 _D	A7 _H	Message contains unknown function code
200 _D	C8 _H	Task not initiated / not configured, data base not found

Table 1-10 continued: Message errors

1.3 FK2 - Serious errors

Error identifier in MW 254,08 / %MW1254.8: 200_D or C8_H resp.

Error class	Error description	Detailed info 1 in MW 254,09 / %MW1254.9		Detailed info 2 in MW 254,10 / %MW1254.10		Detailed info 3 in MW 254,11 / %MW1254.11		Detailed info 4 in MW 254,12 / %MW1254.12		Detailed info 5 in MW 254,13 / %MW1254.13		Detailed info 6 in MW 254,14 / %MW1254.14	
		Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
FK2 Serious error	Error during dynamic memory allocation during loading the 907 AC 1131 PROFIBUS configuration data	0		256 _D	100 _H	8 _D	8 _H	0		0		0	
	Error during dynamic memory allocation during reading/writing of configuration data from/on the SmartMedia Card	0		256 _D	100 _H	8 _D	8 _H	0		31 _D	1F _H	0	
	Error during dynamic memory allocation during providing the general driver resources	0		256 _D	100 _H	8 _D	8 _H	0		33 _D	21 _H	0	
	Error during dynamic memory allocation during providing the structure of the output data	0		256 _D	100 _H	8 _D	8 _H	ID		0 _D	0 _H	0	
	Error during dynamic memory allocation during providing the structure of the input data	0		256 _D	100 _H	8 _D	8 _H	ID		1 _D	1 _H	0	
	Error during dynamic memory allocation during providing the message mailboxes	0		256 _D	100 _H	8 _D	8 _H	ID		32 _D	20 _H	0	
	Error during dynamic memory allocation during providing the driver resources	0		256 _D	100 _H	8 _D	8 _H	ID		33 _D	21 _H	0	
	Error during write/read test on coupler DP RAM	ID		256 _D	100 _H	1 _D	1 _H	0		0		0	
	Operating system error of the coupler	ID		512 _D	200 _H	Error no. ¹		0		0		0	

¹ See "Operating system errors and task errors of the coupler"

Table 1-11: FK2 - Serious errors

1.4 FK3 - Light errors

Error identifier in MW 255,00 / %MW1255.0: 200_D or C8_H resp.

Error class	Error description	Detailed info 1 in MW 255,01 / %MW1255.1		Detailed info 2 in MW 255,02 / %MW1255.2		Detailed info 3 in MW 255,03 / %MW1255.3		Detailed info 4 in MW 255,04 / %MW1255.4		Detailed info 5 in MW 255,05 / %MW1255.5		Detailed info 6 in MW 255,06 / %MW1255.6	
		Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
FK3 Light error	Error when testing the coupler watchdog	ID		256 _D	100 _H	2 _D	2 _H	0		0		0	
	Error during loading the configuration data from the SmartMedia Card; Data for not available coupler	ID		256 _D	100 _H	4 _D	4 _H	0		0		0	
	Coupler not found	ID		256 _D	100 _H	4 _D	4 _H	Coupler type ¹		Coupler model ²		0	
	Error during loading the configuration data from the SmartMedia Card; Data for another coupler type	ID		256 _D	100 _H	5 _D	5 _H	0		0		0	
	Incorrect coupler type found	ID		256 _D	100 _H	5 _D	5 _H	Original coupler type ¹		Coupler type found ¹		0	
	Error during loading the configuration data from the SmartMedia Card; Data for another coupler model	ID		256 _D	100 _H	6 _D	6 _H	0		0		0	
	Incorrect coupler model found	ID		256 _D	100 _H	6 _D	6 _H	Original coupler model ²		Coupler model found ²		0	
	Unknown coupler model found	ID		256 _D	100 _H	7 _D	7 _H	0		0		0	
	No coupler reaction to reset	ID		256 _D	100 _H	12 _D	C _H	0		0		0	
	Coupler does not signalize readiness for operation	ID		256 _D	100 _H	13 _D	D _H	0		0		0	
	Configuration data of 907 AC 1131 or SmartMedia Card not valid	ID		256 _D	100 _H	25 _D	19 _H	0		0		0	
	Error during loading the configuration data from the SmartMedia Card; Coupler already has 907 AC 1131 PROFIBUS configuration data	ID		256 _D	100 _H	29 _D	1D _H	0		0		0	
	Error in Task 1 of the coupler	ID		768 _D	300 _H	Error no. ³		0		0		0	
	Error in Task 2 of the coupler	ID		1024 _D	400 _H	Error no. ³		0		0		0	
	Error in Task 3 of the coupler	ID		1280 _D	500 _H	Error no. ³		0		0		0	
	Error in Task 4 of the coupler	ID		1536 _D	600 _H	Error no. ³		0		0		0	
Error in Task 5 of the coupler	ID		1792 _D	700 _H	Error no. ³		0		0		0		

¹ See ,Coupler types'

² See ,Coupler models'

³ See ,Operating system errors and task errors of the coupler'

Table 1-12: FK3 - Light errors

Error class	Error description	Detailed info 1 in MW 255,01 / %MW1255.1		Detailed info 2 in MW 255,02 / %MW1255.2		Detailed info 3 in MW 255,03 / %MW1255.3		Detailed info 4 in MW 255,04 / %MW1255.4		Detailed info 5 in MW 255,05 / %MW1255.5		Detailed info 6 in MW 255,06 / %MW1255.6	
		Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
FK3 Light error	Error in Task 6 of the coupler	ID		2048 _D	800 _H	Error no. ³		0		0		0	
	Error in Task 7 of the coupler	ID		2304 _D	900 _H	Error no. ³		0		0		0	
	Life identifier error of the coupler	ID		2560 _D	A00 _H	10 _D	A _H	0		0		0	
	Message not fetched, therefore deleted	ID		2560 _D	A00 _H	14 _D	E _H	Request identifier ⁴		Request identifier ⁴		0	
	No buffer for request message from configurator available	ID		2560 _D	A00 _H	16 _D	10 _H	0		0		0	
	No buffer for request message from system available	ID		2560 _D	A00 _H	16 _D	10 _H	Request identifier ⁴		Mode ⁴		0	
	No response of the coupler to request message from configurator	ID		2560 _D	A00 _H	17 _D	11 _H	0		0		0	
	No response of the coupler to request message from system	ID		2560 _D	A00 _H	17 _D	11 _H	Request identifier ⁴		Mode ⁴		0	
	Coupler reports error on response of request message	ID		2560 _D	A00 _H	17 _D	11 _H	Request identifier ⁴		Mode ⁴		Error no. ⁵	
	Coupler reports error; Detailed diagnosis via STAT-FB or configurator	ID		2560 _D	A00 _H	30 _D	1E _H	0		0		0	
	Controller in RUN, but coupler has no configuration data	ID		2816 _D	B00 _H	9 _D	9 _H	0		0		0	
	Coupler has 907 AC 1131 configuration as DP master, but cannot be operated as DP master	ID		2816 _D	B00 _H	18 _D	12 _H	0		0		0	
	Coupler has 907 AC 1131 configuration as DP slave, but cannot be operated as DP slave	ID		2816 _D	B00 _H	19 _D	13 _H	0		0		0	
	907 AC 1131 configuration does not fit with the coupler model	ID		2816 _D	B00 _H	25 _D	19 _H	6 _D	6 _H	0		0	
907 AC 1131 configuration data are of unknown type	ID		2816 _D	B00 _H	25 _D	19 _H	7 _D	7 _H	0		0		

¹ See ‚Coupler types‘

² See ‚Coupler models‘

³ See ‚Operating system errors and task errors of the coupler‘

⁴ See ‚Request and response identifiers with messages‘

⁵ See ‚Message errors‘

Table 1-12 continued: FK3 - Light errors

1.5 FK4 Warnings

Error identifier in MW 255,08 / %MW1255.8: 200_D or C8_H resp.

Error class	Error description	Detailed info 1 in MW 255,09 / %MW1255.9		Detailed info 2 in MW 255,10 / %MW1255.10		Detailed info 3 in MW 255,11 / %MW1255.11		Detailed info 4 in MW 255,12 / %MW1255.12		Detailed info 5 in MW 255,13 / %MW1255.13		Detailed info 6 in MW 255,14 / %MW1255.14	
		Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
FK4 Warning	907 AC 1131 incorrect configuration data	0		2816 _D	B00 _H	25 _D	19 _H	0		0		0	
	907 AC 1131 configuration data contain an invalid ID (card number)	invalid ID		2816 _D	B00 _H	26 _D	1A _H	0		0		0	
	907 AC 1131 configuration data contain PROFIBUS master data, but there is no PROFIBUS master coupler inserted in the slot ID	ID		2816 _D	B00 _H	18 _D	12 _H	0		0		0	
	907 AC 1131 configuration data contain PROFIBUS slave data, but there is no PROFIBUS slave coupler inserted in the slot ID	ID		2816 _D	B00 _H	19 _D	13 _H	0		0		0	

Table 1-13: FK4 Warnings

2 Tables

Table 1-1: Structure of the error messages of the internal couplers	1-1
Table 1-2: Detailed info 2 – Error class	1-2
Table 1-3: Detailed info 3 – Cause of errors, general	1-3
Table 1-4: Detailed info 3 – Cause of errors, Operating system errors and task errors of the coupler.....	1-4
Table 1-5: Detailed infos 4 / 5 - Types of couplers	1-6
Table 1-6: Detailed infos 4 / 5 - Coupler models	1-6
Table 1-7: Detailed infos 4 / 5 - Memory errors	1-6
Table 1-8: Detailed infos 4 / 5 - Errors during loading the configuration data	1-6
Table 1-9: Detailed info 4/5 - Request and response identifiers in messages	1-7
Table 1-10: Message errors	1-7
Table 1-11: FK2 - Serious errors.....	1-10
Table 1-12: FK3 - Light errors.....	1-11
Table 1-13: FK4 Warnings.....	1-13

D

- Detailed info 1 – Error messages of the internal couplers 1-1
- Detailed info 2 – Error messages of the internal couplers 1-2
- Detailed info 3 – Error messages of the internal couplers 1-3

E

- Error messages of the internal couplers series 90 1-1

F

- FK2 internal couplers 1-10
- FK3 internal couplers 1-11
- FK4 internal couplers 1-13

L

- Light errors internal couplers - FK3 1-11

S

- Serious errors internal couplers – FK2 1-10
- Structure of the error messages of the internal couplers 1-1

W

- Warnings internal couplers – FK4 1-13
-

Content

1	Importing 907 FB 1131 configuration data into a 907 AC 1131 project	L 1-1
1.1	Introduction	1-1
1.2	General procedure	1-1
1.3	Organization of the coupler operand ranges	1-2
1.4	Accessing the different data formats	1-3
1.4.1	Accessing BOOL data types	1-3
1.4.2	Accessing BYTE data types	1-4
1.4.3	Accessing SINT or USINT data types	1-5
1.4.4	Accessing WORD data types	1-6
1.4.5	Accessing INT or UINT data types	1-7
1.4.6	Accessing DWORD, DINT and UDINT data types	1-8
1.4.7	Accessing REAL or LREAL data types	1-9
1.4.8	Accessing STRING data types	1-9
1.4.9	Accessing TIME, TIME_OF_DAY, DATE and DATE_AND_TIME data types	1-10
1.4.10	Accessing ARRAYS	1-10
1.4.11	Accessing structures (STRUCT)	1-11
1.5	Determining the operands for a 907 AC 1131 project	1-12
1.5.1	Slave coupler	1-12
1.5.2	Master coupler	1-14
2	Figures	2-1
3	Tables	3-1

1 Importing 907 FB 1131 configuration data into a 907 AC 1131 project

1.1 Introduction

The programming software 907 AC 1131 as well as the 907 FB 1131 configuration tool for fieldbus couplers are independent and standalone applications running on a PC. The coupler configuration data created using 907 FB 1131 have to be imported manually into the related 907 AC 1131 project.

1.2 General procedure

First a description of the bus system, in which the control and the appropriate coupler are used as subscribers, has to be created using 907 FB 1131 and then this configuration has to be downloaded to the control (see 907 FB 1131 documentation).

Depending on the configuration, the operands are obtained used to access the process data of the coupler in the user defined program. The operand designations are composed of the data transmission direction, data format, coupler slot number and the data offset. Input data are read using %I operands, output data are written using %Q operands. This prefix is followed by the format of the required data (X means BOOL or bit, B means BYTE and W represents WORD). The following numbers, separated by a dot, represent the slot number of the coupler and the data offset within this operand range. The calculation of the offset depends on the data type (see 907 AC 1131 documentation Operands of the central unit).

Examples:

%QB1.0 output byte (%QB) of the coupler in slot 1 with offset 0 (BYTE)

%IW2.7 input word (%IW) of the coupler in slot 2 with offset 7 (WORD)

%QX1.0.7 output bit (%QX) of the coupler in slot 1 with offset 0 (WORD), 7th bit

1.3 Organization of the coupler operand ranges

The operand ranges assigned to the couplers can be accessed BOOL/BIT, BYTE or WORD wise, as desired. Depending on the coupler and the transmission direction, the following operands are defined:

BOOL X0.0 - X1791.15
 BYTE B0 - B3583
 WORD W0 - W1791

The following overview shows a typical organization of these operands within the output operands range of a coupler in slot 1.

%QW1. 1791															
%QB1. 3583								%QB1. 3582							
%QX1. 1791.15	%QX1. 1791.14	%QX1. 1791.13	%QX1. 1791.12	%QX1. 1791.11	%QX1. 1791.10	%QX1. 1791.9	%QX1. 1791.8	%QX1. 1791.7	%QX1. 1791.6	%QX1. 1791.5	%QX1. 1791.4	%QX1. 1791.3	%QX1. 1791.2	%QX1. 1791.1	%QX1. 1791.0
%QW1. 1790															
%QB1. 3581								%QB1. 3580							
%QX1. 1790.15	%QX1. 1790.14	%QX1. 1790.13	%QX1. 1790.12	%QX1. 1790.11	%QX1. 1790.10	%QX1. 1790.9	%QX1. 1790.8	%QX1. 1790.7	%QX1. 1790.6	%QX1. 1790.5	%QX1. 1790.4	%QX1. 1790.3	%QX1. 1790.2	%QX1. 1790.1	%QX1. 1790.0
.															
.															
.															
%QW1. 1															
%QB1. 3								%QB1. 2							
%QX1. 1.15	%QX1. 1.14	%QX1. 1.13	%QX1. 1.12	%QX1. 1.11	%QX1. 1.10	%QX1. 1.9	%QX1. 1.8	%QX1. 1.7	%QX1. 1.6	%QX1. 1.5	%QX1. 1.4	%QX1. 1.3	%QX1. 1.2	%QX1. 1.1	%QX1. 1.0
%QW1. 0															
%QB1. 1								%QB1. 0							
%QX1. 0.15	%QX1. 0.14	%QX1. 0.13	%QX1. 0.12	%QX1. 0.11	%QX1. 0.10	%QX1. 0.9	%QX1. 0.8	%QX1. 0.7	%QX1. 0.6	%QX1. 0.5	%QX1. 0.4	%QX1. 0.3	%QX1. 0.2	%QX1. 0.1	%QX1. 0.0

Table 1-1 Organization of the operands range of the couplers

1.4 Accessing the different data formats

1.4.1 Accessing BOOL data types

If the fieldbus I/Os contain data which have to be interpreted as BOOL, these data can be accessed in different ways.

Direct access to operands

Data of the type BOOL can be read or written by directly accessing the related operands.

Example:

A fieldbus device transmits four BOOL values to the control, which are stored beginning at %IX1.2.0. These data should be accessed directly using operands. The appropriate operands are addressed as follows:

```
%IX1.2.0  %IX1.2.1  %IX1.2.2  %IX1.2.3
```

Access using symbolic variables

Data of the type BOOL can also be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four BOOL values to the control, which are stored beginning at %IX1.2.0. These data should be accessed using symbolic variables. The variables declarations could be as follows:

```
VAR
  Var1 AT %IX1.2.0 : BOOL;
  Var2 AT %IX1.2.1 : BOOL;
  Var3 AT %IX1.2.2 : BOOL;
  Var4 AT %IX1.2.3 : BOOL;
END_VAR
```

It is also possible to combine the variables in an ARRAY or an user defined STRUCT.

```
TYPE StructureType:
```

```
STRUCT
  Var1 : BOOL;
  Var2 : BOOL;
  Var3 : BOOL;
  Var4 : BOOL;
```

```
END_STRUCT
END_TYPE
```

```
VAR
  Var AT %IB1.4 : StructureType;
END_VAR
```

or

```
VAR
  Var AT %IB1.4 : ARRAY [1..4] OF BOOL;
END_VAR
```

1.4.2 Accessing BYTE data types

If the fieldbus I/Os contain data which have to be interpreted as BYTE, these data can be accessed in different ways.

Direct access to operands

Data of the type BYTE can be read or written by directly accessing the related operands.

Example:

A fieldbus device transmits four BYTE values to the control, which are stored beginning at %IB1.4. These data should be accessed directly using operands. The appropriate operands are addressed as follows:

```
%IB1.4    %IB1.5    %IB1.6    %IB1.7
```

Access using symbolic variables

Data of the type BYTE can also be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four BYTE values to the control, which are stored beginning at %IB1.4. These data should be accessed using symbolic variables. The variables declarations could be as follows:

```
VAR
  Var1 AT %IB1.4 : BYTE;
  Var2 AT %IB1.5 : BYTE;
  Var3 AT %IB1.6 : BYTE;
  Var4 AT %IB1.7 : BYTE;
END_VAR
```

It is also possible to combine the variables in an ARRAY or an user defined STRUCT.

```
TYPE StructureType:
```

```
STRUCT
  Var1 : BYTE;
  Var2 : BYTE;
  Var3 : BYTE;
  Var4 : BYTE;
```

```
END_STRUCT
END_TYPE
```

```
VAR
  Var AT %IB1.4 : StructureType;
END_VAR
```

or

```
VAR
  Var AT %IB1.4 :ARRAY [1..4] OF BYTE;
END_VAR
```

1.4.3 Accessing SINT or USINT data types

If the fieldbus I/Os contain data which have to be interpreted as SINT or USINT, these data can only be accessed using symbolic variables.

Access using symbolic variables

Data of the type SINT or USINT can be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four BYTE values to the control, which are stored beginning at %IB1.4. These data should be interpreted as SINT or USINT. The variables declarations could be as follows:

```
VAR
  Var1 AT %IB1.4 : SINT;
  Var2 AT %IB1.5 : SINT;
  Var3 AT %IB1.6 : USINT;
  Var4 AT %IB1.7 : USINT;
END_VAR
```

It is also possible to combine the variables in an ARRAY or an user defined STRUCT.

TYPE StructureType:

```
STRUCT
  Var1 : SINT;
  Var2 : SINT;
  Var3 : USINT;
  Var4 : USINT;
```

```
END_STRUCT
END_TYPE
```

```
VAR
  Var AT %IB1.4 : StructureType;
END_VAR
```

or

```
VAR
  Var12 AT %IB1.4 :ARRAY [1..2] OF SINT;
  Var34 AT %IB1.6 :ARRAY [1..2] OF USINT;
END_VAR
```

1.4.4 Accessing WORD data types

If the fieldbus I/Os contain data which have to be interpreted as WORD, these data can be accessed in different ways.

Direct access to operands

Data of the type WORD can be read or written by directly accessing the related operands.

Example:

A fieldbus device transmits two WORD values to the control, which are stored beginning at %IW1.2. These data should be accessed directly using operands. The appropriate operands are addressed as follows:

```
%IW1.2    %IW1.3
```

Access using symbolic variables

Data of the type WORD can also be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four WORD values to the control, which are stored beginning at %IW1.2. These data should be accessed using symbolic variables. The variables declarations could be as follows:

```
VAR
  Var1 AT %IW1.2 : WORD;
  Var2 AT %IW1.3 : WORD;
  Var3 AT %IW1.4 : WORD;
  Var4 AT %IW1.5 : WORD;
END_VAR
```

It is also possible to combine the variables in an ARRAY or an user defined STRUCT.

```
TYPE StructureType:
```

```
STRUCT
  Var1 : WORD;
  Var2 : WORD;
  Var3 : WORD;
  Var4 : WORD;
```

```
END_STRUCT
END_TYPE
```

```
VAR
  Var AT %IW1.2 : StructureType;
END_VAR
```

or

```
VAR
  Var AT %IW1.2 :ARRAY [1..4] OF WORD;
END_VAR
```

1.4.5 Accessing INT or UINT data types

If the fieldbus I/Os contain data which have to be interpreted as INT or UINT, these data can only be accessed using symbolic variables.

Access using symbolic variables

Data of the type INT or UINT can be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four WORD values to the control, which are stored beginning at %IW1.2. These data should be interpreted as INT or UINT. The variables declarations could be as follows:

```
VAR
  Var1 AT %IW1.2 : INT;
  Var2 AT %IW1.3 : INT;
  Var3 AT %IW1.4 : UINT;
  Var4 AT %IW1.5 : UINT;
END_VAR
```

It is also possible to combine the variables in an ARRAY or an user defined STRUCT.

```
TYPE StructureType:
STRUCT
  Var1 : INT;
  Var2 : INT;
  Var3 : UINT;
  Var4 : UINT;
END_STRUCT
END_TYPE

VAR
  Var AT %IW1.2 : StructureType;
END_VAR
```

or

```
VAR
  Var12 AT %IW1.2 :ARRAY [1..2] OF INT;
  Var34 AT %IW1.4 :ARRAY [1..2] OF UINT;
END_VAR
```

1.4.6 Accessing DWORD, DINT and UDINT data types

If the fieldbus I/Os contain data which have to be interpreted as DWORD, DINT or UDINT, these data can only be accessed by declaring ARRAYS or STRUCTS.

Access using symbolic variables

Data of the type DWORD, DINT or UDINT can be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four WORD values to the control, which are stored beginning at %IW1.2. These data should be interpreted as DINT or UDINT. The variables declarations could be as follows:

```
TYPE StructureType:
STRUCT
    Var1 : DINT;
    Var2 : UDINT;
END_STRUCT
END_TYPE

VAR
    Var AT %IW1.2 : StructureType;
END_VAR
```

or

```
VAR
    Var1 AT %IW1.2 :ARRAY [1..1] OF DINT;
    Var2 AT %IW1.4 :ARRAY [1..1] OF UDINT;
END_VAR
```


1.4.7 Accessing REAL or LREAL data types

If the fieldbus I/Os contain data which have to be interpreted as REAL or LREAL, these data can only be accessed by declaring ARRAYS or STRUCTs.

Access using symbolic variables

Data of the type REAL or LREAL can be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four WORD values to the control, which are stored beginning at %IW1.2. These data should be interpreted as REAL or LREAL. The variables declarations could be as follows:

```
TYPE StructureType:
STRUCT
    Var1 : REAL;
    Var2 : REAL;
    Var3 : LREAL;
END_STRUCT
END_TYPE

VAR
    Var AT %IW1.2 : StructureType;
END_VAR
```

or

```
VAR
    Var12 AT %IW1.2 :ARRAY [1..2] OF REAL;
    Var3 AT %IW1.4 :ARRAY [1..1] OF LREAL;
END_VAR
```

1.4.8 Accessing STRING data types

If the fieldbus I/Os contain data which have to be interpreted as STRING, these data can only be accessed by declaring STRING variables.

Access using symbolic variables

Data of the type STRING can be accessed by declaring appropriate variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits four BYTE values to the control, which are stored beginning at %IB1.4. The data should be interpreted as STRING. The variables declarations could be as follows:

```
VAR
    Var1 AT %IB1.4 :STRING(4);
END_VAR
```

1.4.9 Accessing TIME, TIME_OF_DAY, DATE and DATE_AND_TIME data types

If the fieldbus I/Os contain data which have to be interpreted as TIME, TIME_OF_DAY, DATE or DATE_AND_TIME, these data can only be accessed by declaring ARRAYS or STRUCTs.

Access using symbolic variables

Data of the type TIME, TIME_OF_DAY, DATE or DATE_AND_TIME can be accessed by declaring appropriate variables, which are located at the related operand addresses. Each variable of these types allocates 2 words in the memory.

Example:

A fieldbus device transmits eight WORD values to the control, which are stored beginning at %IW1.2. The data should be interpreted as TIME, TIME_OF_DAY, DATE or DATE_AND_TIME. The variables declarations could be as follows:

```
TYPE StructureType:
STRUCT
    Var1 : TIME;
    Var2 : TIME_OF_DAY;
    Var3 : DATE;
    Var4 : DATE_AND_TIME;
END_STRUCT
END_TYPE

VAR
    Var AT %IW1.2 : StructureType;
END_VAR
```

or

```
VAR
    Var1 AT %IW1.2 :ARRAY [1..1] OF TIME;
    Var2 AT %IW1.4 :ARRAY [1..1] OF TIME_OF_DAY;
    Var3 AT %IW1.6 :ARRAY [1..1] OF DATE;
    Var4 AT %IW1.8 :ARRAY [1..1] OF DATE_AND_TIME;
END_VAR
```

1.4.10 Accessing ARRAYS

If fieldbus I/Os should be combined in ARRAYS, the appropriate variables, which are located at the related operand addresses, have to be declared.

Example:

A fieldbus device transmits four BYTE values to the control, which are stored beginning at %IB1.4. These data should be accessed as elements of an ARRAY. The variables declarations could be as follows:

```
VAR
    Var AT %IB1.4 :ARRAY [1..4] OF BYTE;
END_VAR
```

1.4.11 Accessing structures (STRUCT)

If fieldbus I/Os should be combined in structures, this is done by declaring the appropriate individual data types and variables, which are located at the related operand addresses.

Example:

A fieldbus device transmits 16 BYTE to the control, which are stored beginning at %IB1.4. The data are formatted as structure with mixed data types. The declaration of an appropriate variable could be as follows:

```
VAR
    Var AT %IB1.4 :ARRAY [1..4] OF BYTE;
END_VAR

TYPE StructureType:
STRUCT
    Var1: WORD;
    Var2: STRING(4);
    Var3: ARRAY [1..4] OF BYTE;
    Var4: WORD;
    Var5: DWORD;
END_STRUCT
END_TYPE

VAR
    Structure AT %IB1.4 : StructureType;
END_VAR
```

1.5 Determining the operands for a 907 AC 1131 project

1.5.1 Slave coupler

In a slave coupler the operands are stored separately for each transmission direction, according to the related I/O configuration. At the beginning always the offset 0 is used. In addition it has to be observed, that word-oriented data can only start at even BYTE offsets.

The determination of the operands is described using an example configuration. For a **slave coupler in slot 2** the following I/O configuration is specified:

No	Size (number of data types)	Data type	Transmission direction	Process direction
1	2	BYTE	Input	Output
2	1	BYTE	Output	Input
3	1	WORD	Input	Output
4	2	WORD	Output	Input

Table 1-2: Specification of I/O configuration for a slave coupler in slot 2

First it has to be observed that the classification of a (slave) device as input or output device is usually done from the process point of view. Thus, an input device reads input data from the process and outputs them via the bus. If input data are mentioned in the following description, this refers to data which can be received via the bus. Output data are data which are transmitted to the master via the bus.

As already described above, the first date in each direction always starts with offset 0. Since the first fieldbus input date (no. 1) is byte-oriented with the size of 2 BYTE, the resulting operands are **%IB2.0** for the first and **%IB2.1** for the second BYTE of the module. The next fieldbus input date (no. 3) is a word. Within the operand range the first two BYTES are already allocated by the preceding module. Due to this, the operands **%IB2.2** and **%IB2.3** are reserved for the WORD module. This is equal to **%IW2.1** (see 907 AC 1131 documentation Operands of the central unit). Since this is a word-oriented date, it is usually accessed in the project with **%IW2.1**.

Analogue to the input data the fieldbus outputs start with offset 0. For that reason module no. 2 occupies operand **%QB2.0**. The following output module would normally start with operand %QB2.1. But since this is a WORD module, it has to start with an operand having an even BYTE offset. This means that operand %QB2.1 must not be used. Instead, the operands %QB2.2 and %QB2.3 (correspond to **%QW2.1**) are used for the first WORD of the module and the operands %QB2.4 and %QB2.5 (correspond to **%QW2.2**) are used for the second WORD of the module.

Due to this, the following variables have to be declared for the 07 KT 97 DPS slave in the 907 AC 1131 project:

```
VAR
  (* variables of the 07 KT 97 DPS *)
  KT97DPS_Eingabe_Byte_1 AT %IB2.0 : BYTE;      (* IB 0 *)
  KT97DPS_Eingabe_Byte_2 AT %IB2.1 : BYTE;      (* IB 1 *)
  KT97DPS_Eingabe_Wort_1 AT %IW2.1 : WORD;       (* IB 2 and IB 3 *)
  KT97DPS_Ausgabe_Byte_1 AT %QB2.0 : BYTE;      (* QB 0 *)
  KT97DPS_Ausgabe_Wort_1 AT %QW2.1 : WORD;      (* QB 2 and QB 3 *)
  KT97DPS_Ausgabe_Wort_2 AT %QW2.2 : WORD;      (* QB 4 and QB 5 *)
END_VAR
```

1.5.2 Master coupler

If automatic addressing for a master coupler is activated (see 907 FB 1131 documentation), the I/O data from the slaves are stored directly one behind the other. If auto-addressing is disabled, the data can be stored at any location within the given address range. In both cases the resulting data distribution can be displayed in 907 FB 1131 using the menu item **View – Address table**. This table also supports you when determining the related operands for 907 AC 1131 projects.

The following example shows the address table (sorted according to the data addresses) of a **master coupler** with **two assigned slaves**, which are both configured as described in the above example.

Addr.	Slot	Idx.	Device	Module	Symbol name	IType	I Addr.	I Len.	OType	O Addr.	O Len.
2	0	1	07 KT 97-DPS	2 x 8 bit input	Module1	IB	0	2			
2	1	1	07 KT 97-DPS	1 word input	Module2	IW	2	1			
3	0	1	07 KT 97-DPS	2 x 8 bit input	Module1	IB	4	2			
3	1	1	07 KT 97-DPS	1 word input	Module2	IW	6	1			
2	2	1	07 KT 97-DPS	1 x 8 bit output	Module3				QB	0	1
2	3	1	07 KT 97-DPS	2 word output	Module4				QW	1	2
3	2	1	07 KT 97-DPS	1 x 8 bit output	Module3				QB	5	1
3	3	1	07 KT 97-DPS	2 word output	Module4				QW	6	2

Figure 1-1 Address table

First it has to be observed that the classification of a (slave) device as input or output device is usually done from the process point of view. Thus, an input device reads input data from the process and outputs them via the bus. If input data are mentioned in the following description, this refers to data which the master receives from a slave via the bus. Output data are data which are transmitted from the master to a slave via the bus.

To determine the operands, only the columns **I Type** (input data type), **I Addr.** (input data offset), **I Sz.** (input data size) and the corresponding output data columns are relevant. The columns Addr. (bus address), Slot (module), Device (device type), Module (module type) and Symb. Name (symbolic module name) are only used to assign the determined operands to the corresponding module of the corresponding device in the program. In addition it is assumed, that the **master coupler** is located **in the control in slot 2**.

The columns I Addr. and O Addr. always describe the BYTE offset within the corresponding address range; the columns E Sz. and O Sz. indicate the size of the module, depending on the data type.

If the input data are regarded first, the assigned operands result as follows:

The first input module (line 1) is a BYTE input module (IB) with the offset 0 (BYTE) and the size 2 (BYTE). For this module the operands **%IB2.0** and **%IB2.1** result. Therefore, the next input module shown in line 2 gets the operands **%IB2.2** and **%IB2.3**. Since these data are word-based data, they are usually accessed using **%IW2.1**. Analogue to this, the input modules of the slave with bus address 3 allocate the operands **%IB2.4** and **%IB2.5** and **%IB2.6** / **%IB2.7** and **%IW2.3** respectively.

The first output module (line 5) is byte-oriented with the offset 0 (BYTE) and the size 1 (BYTE); thus it allocates the operand **%QB2.0**. According to this systematic, the next output module in line 6 should start with operand **%QB2.1**. Since this is a module with word-oriented data, it has to start with an operand having an even BYTE offset. The next higher even BYTE offset is 2. Therefore, the WORD module allocates the operands **%QB2.2** and **%QB2.3** or **%QW2.1** respectively and **%QB2.4** and **%QB2.5** or **%QW2.2** respectively. The operands for the last two modules are derived in the same way, again with a 1 BYTE offset for the last module (**%QB2.6**, **%QW2.4** and **%QW2.5**).

According to this, the following variables have to be declared in the 907 AC 1131 project:

VAR

(* Variables of: 07 KT 97 DPS with address 2 *)

KT97DPS_2_Input_Byte_1 AT %IB2.0 : BYTE;	(* IB 0 *)
KT97DPS_2_Input_Byte_2 AT %IB2.1 : BYTE;	(* IB 1 *)
KT97DPS_2_Input_Word_1 AT %IW2.1 : WORD;	(* IB 2 and IB 3 *)
KT97DPS_2_Output_Byte_2 AT %QB2.0 : BYTE;	(* QB 0 *)
KT97DPS_2_Output_Word_1 AT %QW2.1 : WORD;	(* QB 2 and QB 3 *)
KT97DPS_2_Output_Word_2 AT %QW2.2 : WORD;	(* QB 4 and QB 5 *)

(* Variables of: 07 KT 97 DPS with address 3 *)

KT97DPS_2_Input_Byte_1 AT %IB2.4 : BYTE;	(* IB 4 *)
KT97DPS_2_Input_Byte_2 AT %IB2.5 : BYTE;	(* IB 5 *)
KT97DPS_2_Input_Word_1 AT %IW2.3 : WORD;	(* IB 6 and IB 7 *)
KT97DPS_2_Output_Byte_1 AT %QB2.6 : BYTE;	(* QB 6 *)
KT97DPS_2_Output_Word_1 AT %QW2.4 : WORD;	(* QB 8 and QB 9 *)
KT97DPS_2_Output_Word_2 AT %QW2.5 : WORD;	(* QB 10 and QB 11 *)

END_VAR

2 Figures

Figure 1-1 Address table	1-14
--------------------------------	------

3 Tables

Table 1-1 Organization of the operands range of the couplers.....1-2
Table 1-2: Specification of I/O configuration for a slave coupler in slot 21-12

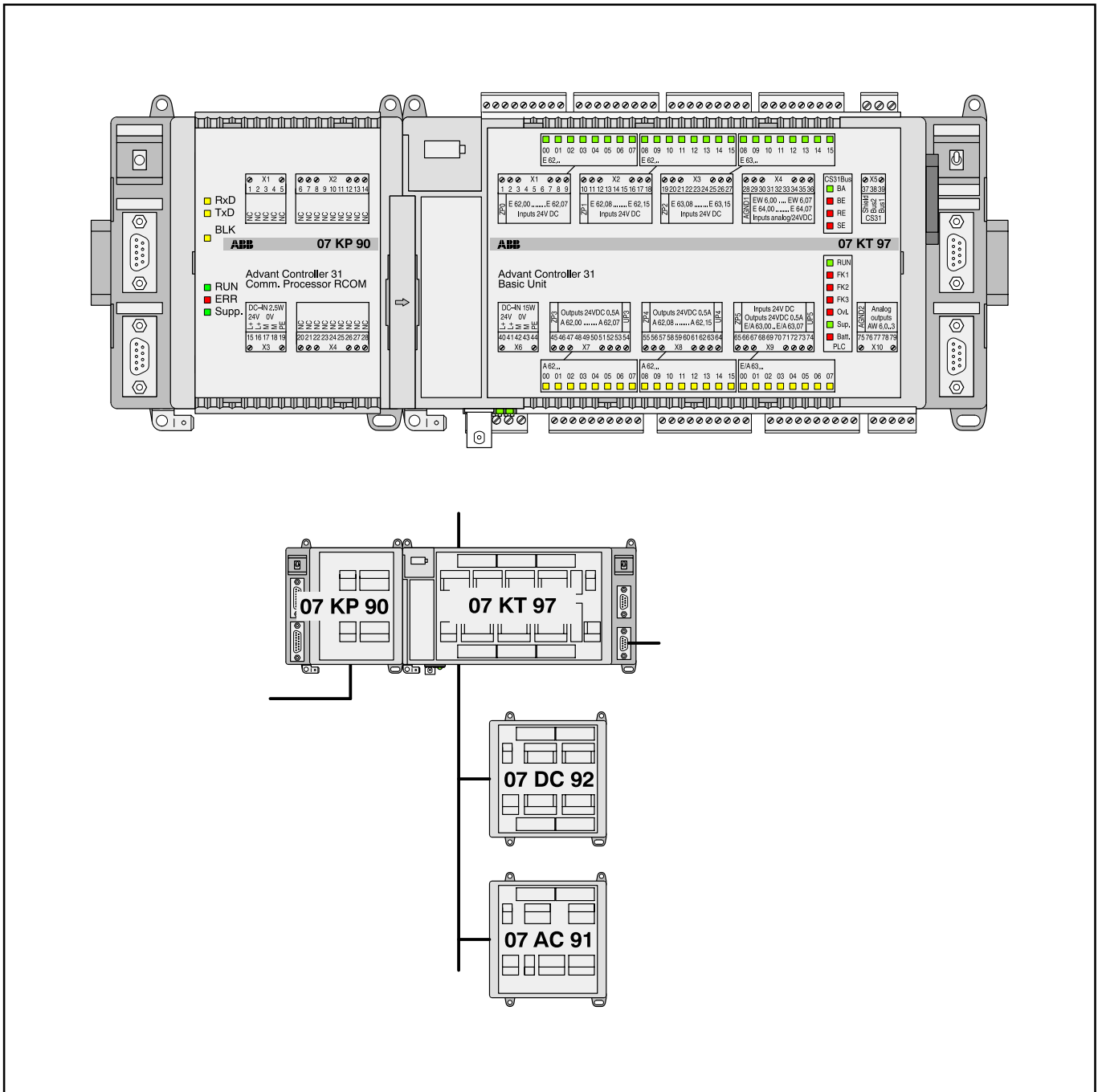


ABB STOTZ-KONTAKT GmbH

Eppelheimer Straße 82 Postfach 101680
69123 Heidelberg 69006 Heidelberg
Germany Germany

Telephone +49 6221 701-0
Telefax +49 6221 701-240
E-Mail desst.help@de.abb.com
Internet <http://www.abb.de/stotz-kontakt>

System Technology 90 Series External Couplers



Content

1	07 KP 90 - Coupler for RCOM/RCOM+	1-1
1.1	Features of the module 07 KP 90 R303	1-1
1.2	Communication via RCOM/RCOM+	1-1
1.2.1	List of function blocks for RCOM/RCOM+	1-2
1.2.2	Program examples for RCOM communication	1-2
1.3	Introduction to RCOM	1-3
1.3.1	What is RCOM?	1-3
1.3.2	Data transmission	1-4
1.3.3	Job types	1-6
1.3.4	Addressing in the RCOM network	1-7
1.3.5	Overview of the 07 KP 90 R303	1-8
1.3.6	Differences between RCOM and RCOM+	1-9
1.3.7	Switching between RCOM and RCOM+	1-9
1.4	Planning	1-10
1.4.1	Overview	1-10
1.4.2	Structure of the data sets	1-14
1.4.3	Use of the system services	1-16
1.4.4	The RCOM system time	1-17
1.4.5	Planning procedure	1-17
1.4.6	Hardware handshake	1-18
1.4.7	Special modems	1-21
1.4.8	Behavior in the case of RUN/STOP and Single Step/Single Cycle	1-23
1.5	Using dial-up modems	1-24
1.5.1	General	1-24
1.5.2	Timeout	1-25
1.5.3	Modem parameters	1-26
1.5.4	Setup	1-26
1.5.5	Telephone numbers	1-28
1.5.6	Entering the files	1-29
1.5.7	Operation with LGH 9600H1	1-30
1.6	Operator	1-33
1.6.1	What is "Operator"?	1-33
1.6.2	Fault-finding: Debug	1-33
1.6.3	Operator commands	1-35
1.7	Error codes	1-37
1.8	Operator messages	1-39
1.9	Instructions for RCOM communication commissioning	1-44
1.9.1	Presetting of the dial-up modems: e. g. ELSA Microlink 14.4TQ	1-44
1.9.2	Connecting modems with the RCOM interface of the 07 KP 90	1-45
1.9.3	Connecting terminal-PC with the Console interface of 07 KP 90	1-46
1.9.4	Parameterization of coupler 07 KP 90	1-46

2	07 KP 93 - Coupler for MODBUS RTU	2-1
2.1	Features of the module 07 KP 93 R1163	2-1
2.2	Communication via the MODBUS interfaces COM3 and COM4	2-1
2.3	List of function blocks for 07 KP 93	2-1
3	Communication module 07 MK 92 R0161	3-1
3.1	Features of the communication 07 MK 92 R1161	3-1
3.2	Notes concerning the programming software 907 MK 92	3-2
3.3	List of function blocks for the communication with the basic unit 07 MK 92	3-2
4	Figures	4-1
5	Tables	5-1
6	Index	I

1.1 Features of the module 07 KP 90 R303

- The module can be planned as RCOM master or as RCOM slave.
- Up to 254 RCOM slaves are possible in a network (max. 8 slaves when using MasterPiece 200 and max. 30 slaves when using dial-up operation).
- The RCOM protocol is compatible to MP200/1 with DSCA 180A. All RCOM services are available (cold start, warm start, normalization, clock synchronization, writing and reading data, event polling).
- The RCOM interface for the connection of the modem corresponds to EIA RS-232. In addition an operation according to EIA RS-485 is possible.
- An additional user interface (CONSOLE) according to EIA RS-232 is available as a commissioning utility (display of the course of communication, planning of phone numbers, etc.).
- Software clock. This time can be used by the PLC program.

1.2 Communication via RCOM/RCOM+

The communication between the basic unit 07 KT 9x and the coupler 07 KP 90 R303 is performed via the networking interface using function blocks of the programming software 907 AC 1131 (see 907 AC 1131 part 2 volume 9).

The programming and test software 907 KP 90 R303 is not required to perform the programming of the basic units 07 KT 95, 07 KT 96, 07 KT 97, and 07 KT 98. The required function blocks are part of the ABB libraries in the programming software 907 AC 1131.

1.2.1 List of function blocks for RCOM/RCOM+

The function blocks are part of the programming software 907 AC 1131.

Library: RCOM_V40.LIB or RCOM_S90_V41.LIB

The communication between the basic units 07 KT 9x and 07 KP 90 R303 is performed using the following function blocks:

- CLOCK Set clock
- COLDST Cold start
- DIAL Dial communication partner
- HANGUP Hang up phone
- NORMAL Normalization
- POLL Perform event polling
- RCOM Initialize 07 KP 90 R303 for RCOM protocol
- RCOM_PL Initialize 07 KP 90 R303 for RCOM+ protocol
- READ Read data from RCOM slave
- READ_S Provide data for READ job
- RECV Receive data from RCOM partner
- SYS_S RCOM system service
- TRANSM Send data to RCOM partner
- WARMST Warm start

1.2.2 Program examples for RCOM communication

When installing the 907 AC 1131 example projects for RCOM dedicated line and RCOM dial-up line are copied into the directory ..\AC1131\Projekte\Beispiele_S90\RCOM_S90.

1.3 Introduction to RCOM

This chapter explains the fundamental terms and functions of the RCOM protocol. It also briefly presents the functions of the 07 KP 90 R303.

1.3.1 What is RCOM?

RCOM is a transmission protocol which is particularly suitable for data transmission over large distances (RCOM = Remote COMMunication).

The protocol is based upon a simple V.24 interface so as to permit the use of standard data teletransmission devices (e.g. modems).

The main fields of application for RCOM couplings are as follows:

- Coupling from Advant Controller 31 to ABB MasterPiece control systems
- Networking Advant Controller 31 to ABB Procontic T200 controllers with 07 KP 64
- Inter-networking AC31 stations

You can use either dedicated lines (e.g. existing cable paths or leased lines) or telephone lines with dial-up modems for communication.

RCOM networks

An RCOM network consists of two or more users, e.g. control computers or substations etc. One user is always planned as the RCOM master. All other users are RCOM slaves.

The users are interconnected by means of a transmission medium. In the case of RCOM, this may comprise direct lines for instance (point-to-point connections), dedicated lines with multidrop modems (these permit coupling of several users to one line) or dial-up connections via the public telephone network.

Each user has an address via which it can be addressed. This address is a number between 1 and 254 for slaves, and the master has the address 0.

The illustration below shows an RCOM network with multidrop modems:

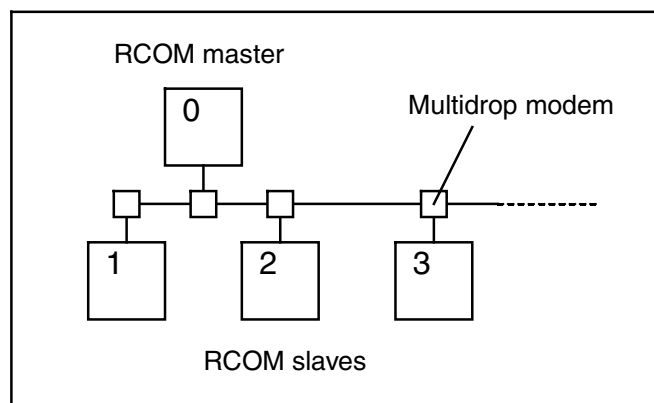


Figure 1-1 RCOM network with multidrop modems

With each job, the master transfers the address of the addressed slave, and only this slave will respond to the job.

Master-slave structure

There is a simple convention for controlling data communication on the line: One user in the network is the **master**. Only the master can send jobs to other users, the **slaves**. The slaves respond to a job telegram with a telegram which indicates whether the job has been understood and whether it has been possible to execute it.

This always results in the sequence job telegram → response telegram on the data line.

1.3.2 Data transmission

Data sets

All user data in the RCOM network are transmitted in the form of so-called data sets (DSs for short). They consist of maximum sixteen 16-bit data words (eight 32-bit double words in the case of MasterPiece).

A maximum of one data set can be transferred in each job telegram. If it is intended to transmit large quantities of data, it is necessary to start several jobs.

For identification, each data set has a number which is also transferred in the telegram. This data set number is abbreviated to 'IDT'.

These data sets are stored in word flags in the AC31. Thus, 16 word flags in which the user data are to be stored must be reserved for one data set.

The user must specify the data set number ('IDT'), the first of 16 word flags (start word flag 'AMW') and the number of data words to be transferred ('LEN') at the corresponding connection element for transfer of the data set.

There are three options for transmission of data sets:

- Write data sets to slave
- Read data sets from slave
- Read data sets from slave.

Write data set

The master can write a data set to the slave by reading the user data from the planned flags and sending them via telegram to the slave. The data are stored in the planned flags in the slave. The slave confirms reception of the data in the response telegram.

Read data set

The master reads a data set from the slave by first sending a job telegram to the slave. The slave receives this telegram, reads the data from its flags and returns the data in the response telegram. The data from the response are stored in the planned flags in the master.

Event polling

In many applications, it is necessary for the slave to transfer data to the master of its own accord, e.g. if it has recognized an important event in the process.

Normally, the slave cannot start communication of its own accord (master–slave structure). So the slave would have to wait until the master reads the required data set.

The RCOM protocol has a simple mechanism for overcoming this problem: Event polling.

If a slave wishes to send a data set to the master of its own accord, it can do this as follows:

- The slave transfers the data set to a queue (event queue) on the 07 KP 90 R303.
- The master cyclically polls all slaves consecutively in order to establish whether they have events in their queues. If so, the addressed slave sends the data set in the response. If not, it sends the response 'Event queue empty'.

This means that the slave can signal events to the master very easily. If no events have occurred, no user data are exchanged either. So the transmission will be completed very quickly in this case.

Since there is no temporal relationship between triggering of the event (insertion in the queue) and event polling (read–out of the queue by the master), a time stamp which provides information on when the event occurred is stored in the data set. For this reason, only maximum 14 data words can be entered in the data set, and the last two words contain the time stamp.

The 07 KP 90 R303 can store a maximum of 20 events in the queue. Other events are rejected with an error message.

The illustration below is intended to clearly show event driven transmission.

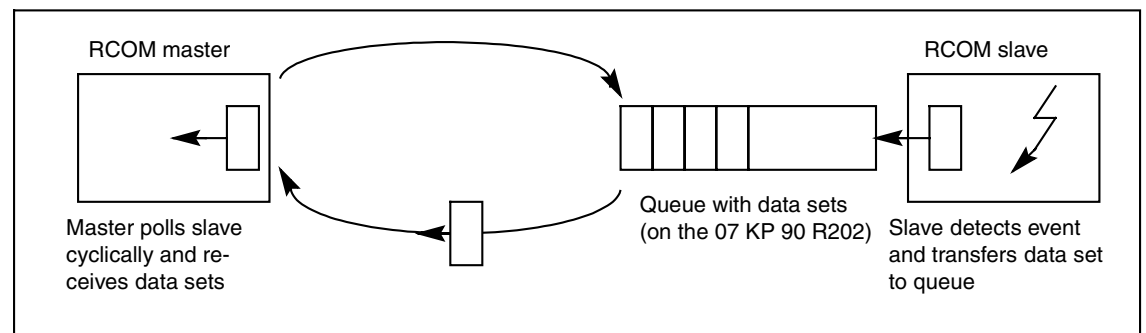


Figure 1-2 Event polling

1.3.3 Job types

There are several job types which can be subdivided into two groups for data transmission and for management of the RCOM network:

- System services, e.g. for connection set-up and cleardown and reinitializing the network etc.
- Services for data transmission. After the RCOM network has been started successfully, the master triggers these jobs in order to transmit user data.

System services

The following services are provided for management of the RCOM network:

- Cold start: The addressed slave is reset, i.e. all protocol control characters are set to initial state. In addition, the contents of the event queue are deleted. After a cold start, data sets cannot be transmitted again until normalization has occurred.
- Warm start: This also deletes the contents of the event queue. A warm start can be started after a transmission error, for instance, in order to resynchronize master and slave. After a warm start, data sets cannot be transmitted again until after normalization.
- Normalization (normalize user part): This enables transmission of data sets after a cold start or warm start. This job **must** be used in order to permit communication to commence.
- Set clock (clock synchronization): The 07 KP 90 R303 has a clock which generates the time stamps for events. This clock can be set by the master.

All system services can be started in the master with corresponding connection elements. In the slave, the system services are handled automatically by the 07 KP 90 R303, i.e. nothing needs to be planned for the system services in the RCOM slaves.

Please do not confuse the RCOM services Cold start and Warm start with the commands of the same name on the AC31 PLC. The terms Cold start and Warm start always refer to the RCOM services in this manual.

Data transmission services

The following services are provided for data transmission:

- Write data set
- Read data set
- Poll event queue (event polling).

1.3.4 Addressing in the RCOM network

Addressing in the RCOM network

A complete address must be specified in order to address a data set on a specific slave. This address consists of the following parts:

- Number of the RCOM network (NET). Since only one RCOM network can be connected on the 07 KP 90 R303, NET must always be zero.
- Number of the slave (NODE). Up to 254 slaves may be present in an RCOM network. The number zero is used by the master. NODE is abbreviated to NOD in the connection elements used for planning the coupler.
- Number of the data set (IDT). IDENT is abbreviated to IDT in the connection elements.
- Number of data words of the data set to be transmitted (LEN). A data set does not always need to be transmitted completely but at least two data words must be transmitted and the number of data words to be transmitted must be an even number. Transmission always commences with the first data word.

The illustration below shows an example of an RCOM system and the addressing path:

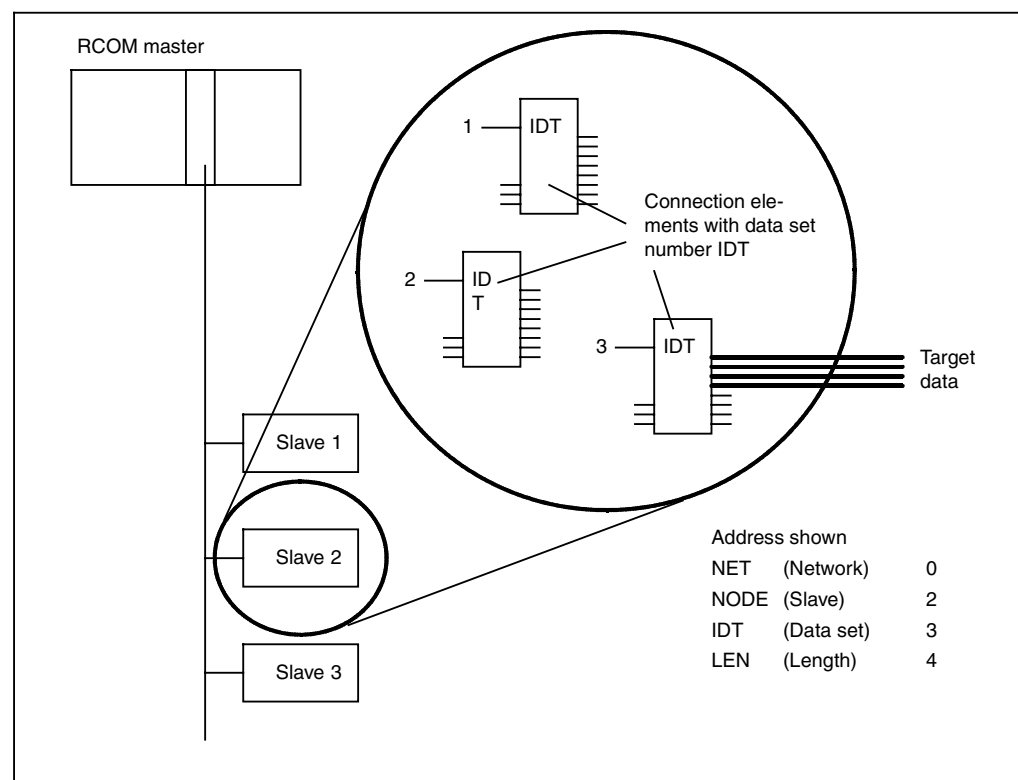


Figure 1-3 Addressing in the RCOM system

Broadcast

Slave number FF H (255) can be used to transmit specific services to all slaves simultaneously. Such a service is **not** answered by **any of the slaves** and is thus repeated several times by the master as a safety measure.

Broadcast is permitted only for job types in which no user data are transmitted (system services).

1.3.5 Overview of the 07 KP 90 R303

Control of the coupler

The coupler is controlled by the AC31 master station with the aid of commands. Commands from the master station and responses of the coupler are exchanged via the networking interface of the AC31 master station.

The user controls the coupler only with the aid of the supplied connection elements of the RCOM library RCOM_S90_Vxx.LIB. There are connection elements for all system and data transmission services. This permits the user himself to define the required communication sequence very simply.

Parameters

All important parameters (baud rate, timeout times etc.) for the coupler are defined in an initialization connection element (RCOM). They are read once by the coupler during initialization and are then used unchanged until the next initialization.

Parameters for the individual services (e.g. slave address, data set number etc.) are preset directly on the connection element for this service.

Commissioning

A terminal can be connected to the 07 KP 90 R303 in order to simplify commissioning. The coupler then issues messages concerning incoming commands from the PLC, received and transmitted RCOM telegrams and any errors which have occurred.

This function can be deactivated after commissioning. The communication processor then continues operation without a terminal.

The Annex provides a list of all messages of the coupler and their significance.

1.3.6 Differences between RCOM and RCOM+

There are the following differences concerning the data transmission with RCOM and RCOM+:

- the "BREAK" character of RCOM is replaced by a transmission break of configurable length for RCOM+
- the 8 bit "Exclusive-or-check sum" of RCOM is replaced by a 16 bit CRC16 check sum for RCOM+
- in addition, the variable DIGI_time was added for delaying the transmission when using the DIGILINE modems

1.3.7 Switching between RCOM and RCOM+

Switching over between RCOM and RCOM+ is carried out by configuring the connection elements RCOM or RCOM+.

The connection elements have the same parameters. Internally, the variable RCOM_typ is set to "1" in case of RCOM+. In case of RCOM, RCOM_typ is "0".

The description of the connection elements will be found in part 2.

1.4 Planning

1.4.1 Overview

Connection elements permitting simple use of the coupler are provided for all required services:

- The coupler is initialized with the “RCOM” connection element. It defines all parameters required for RCOM operation.
- For the system services, there are blocks for
 - the Cold start service
 - the Warm start service
 - normalization
 - event polling
 - event polling
 - telephone dialing and
 - telephone hang-up.
- There are the connection elements TRANSM (Transmit data set) and RECV (Receive data set) for writing data sets. These connection elements are also used for event-driven transmission. Connection element “POLL” which triggers polling of a slave must be planned in the master for event polling.
- The RCOM master uses the READ connection element for reading data sets. The addressed slave makes available the data to be read in the READ_S connection element.

Initialization, Cold start and Normalization

The illustration on the next page shows the application of the connection elements RCOM, COLDST and NORMAL. These connection elements must be used for initializing the communication processors and for starting the RCOM protocol.

Each RCOM user is initialized with the RCOM connection element, i.e. the transmission parameters, the network address and the timeout times etc. are defined.

The RCOM master must then perform an RCOM Cold start service (COLDST). Data transmission must now be enabled with normalization (NORMAL). Cold start and normalization are implemented in the example with Broadcast telegrams (NOD = 255) so that all slaves are addressed simultaneously.

The RCOM network is ready for data transfer after the above described procedures.

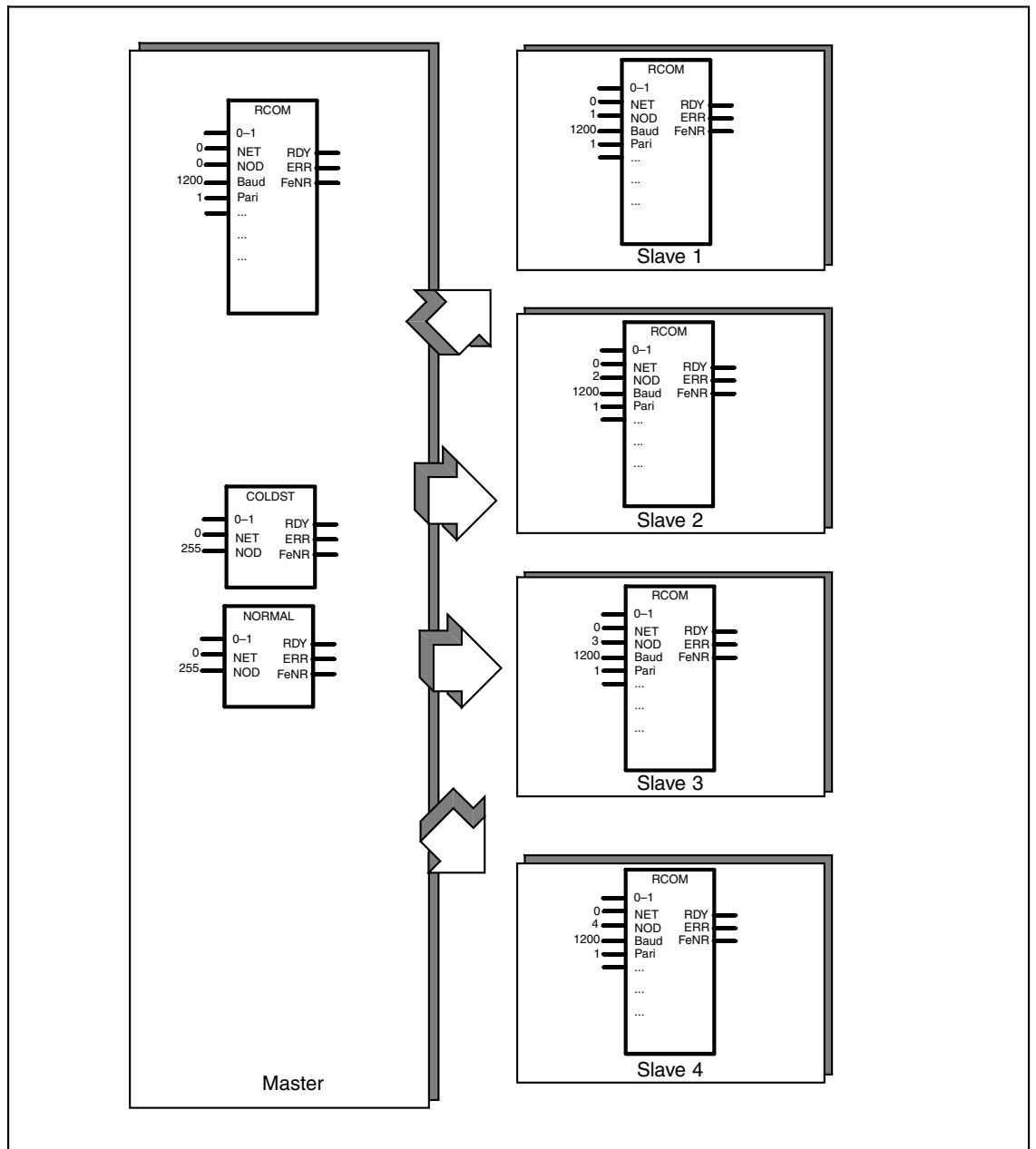


Figure 1-4 Application of the connection elements for initialization

In the application of the connection elements for initialization and the RCOM services Cold start and Normalization the connection elements have been shown in simplified form.

Data transmission

The illustration below shows an example of transmission of data sets. It is intended to clearly illustrate the relationship between connection elements and the significance of address and data set number IDT.

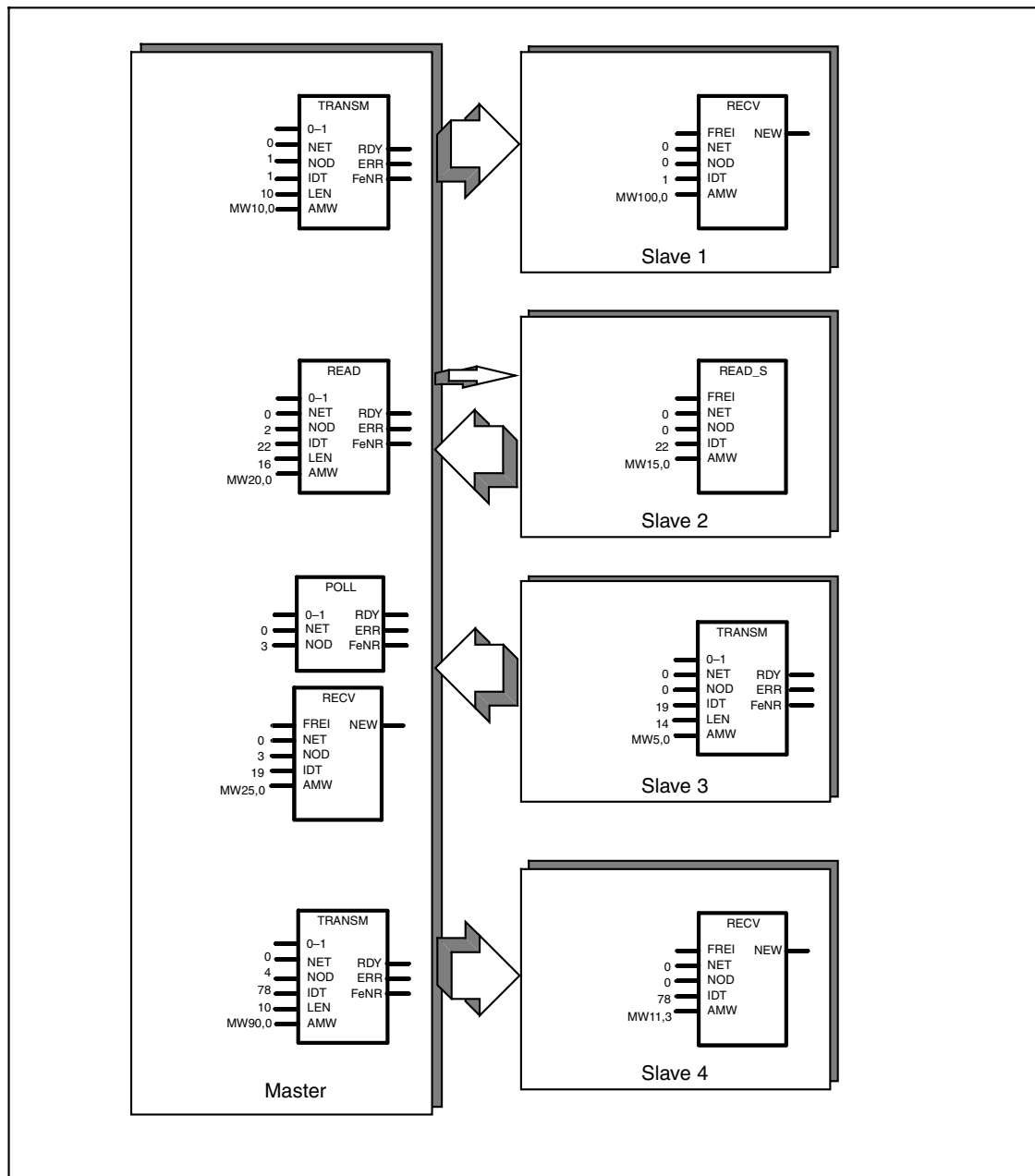


Figure 1-5 Application of the connection elements for a data transmission

In the application of the connection elements for a data transmission the connection elements are shown in simplified form.

The following transmissions can be performed in this example:

- Data set 1 is transmitted from the master to slave 1. The data are taken from MW10,00 / %MW1010.0 to MW10,09 / %MW1010.9 in the master and stored in flags MW100,00 / %MW1100.0 to MW100,09 / %MW1100.9 in the slave.
- Data set 22 is read from slave 2. The data are taken from flags MW15,00 / %MW1015.0 to MW15,15 / %MW1015.15 in the slave and stored in MW20,00 / %MW1020.0 to MW20,15 / %MW1020.15 in the master.
- Data set 19 is sent from slave 3 to the master (event-driven). The master must first poll slave 3 (connection element POLL). The data are fetched from flags MW05,00 / %MW1005.0 to MW05,13 / %MW1005.13 in the slave and are stored in flags MW25,00 / %MW1025.0 to MW25,13 / %MW1025.13 in the master.
- Data set 78 is sent from the master to slave 4. The data are read from MW90,00 / %MW1090.0 to MW90,09 / %MW1090.9 in the master and stored in MW11,03 / %MW1011.3 to MW11,12 / %MW1011.12 in slave 4.

This illustration does not show the logic for enabling etc.

Please note:

- For each connection element in the master, there is a partner in the slave. These two connection elements have the same data set number IDT.
- The actual user data are not applied as inputs or outputs on the connection element, but parameters are assigned to them only as a reference to **where** the data are stored.
- All data sets in the slave have zero as address NOD since jobs can come **only** from the master.

Flag M255,15 / %MX255.15

Flag M255,15 / %MX255.15 requires special handling so that the RCOM connection elements can be initialized correctly when starting the user program.

This flag is always initialized to zero each time the user program is started, i.e. independently of the initialization presets by the system constants.

M255,15 / %MX255.15 must be set to TRUE **at the end of the PLC program**, so that the RCOM connection elements are able to determine whether the user program has been restarted or not. The flag must not be changed after this.

The RCOM connection elements do not work correctly if this rule is not observed.

1.4.2 Structure of the data sets

The data sets must always lie in the normal word flag range MWxx,yy / %MW10xx.yy at a fixed address. This address must be specified at input AMW on the connection elements.

Word flags

Word flags can be transmitted directly. You merely have to transfer the data from the process to the data set, i.e. by means of an assignment or a COPY connection element.

Binary flags

Binary flags must be packed to form words with the PACK connection element for transmission. They can unpacked again accordingly with the UNPACK connection element after transmission.

Double words

Double words must also be separated to form words before transmission. In this case, the high-order word must lie **first** and at an **even address** in the data set (example: data set starts at MW10,00 / %MW1010.0; possible positions for a double word: MW10,00 / %MW1010.0, MW10,02 / %MW1010.2, MW10,04 / %MW1010.4 etc.). This ensures compatibility with MasterPiece control systems.

Example of a data set

The illustration below shows a data set in which various data types are packed:

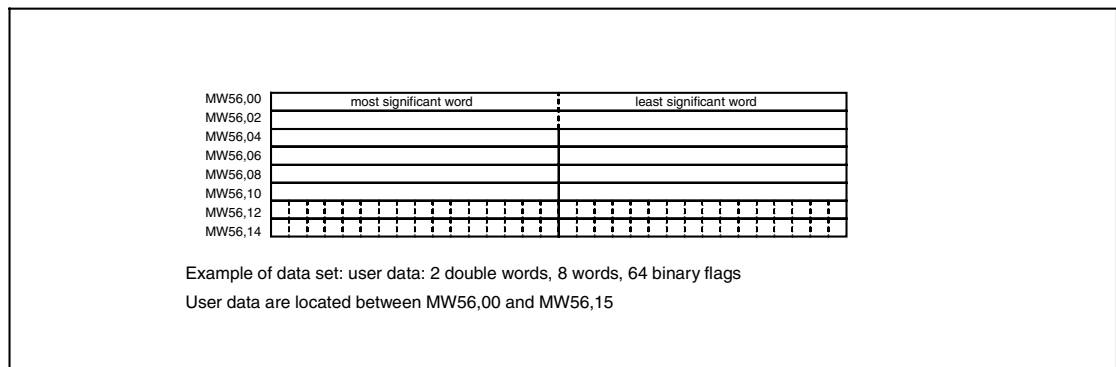


Figure 1-6 Data set in which various data types are packed

Data sets in case of event-driven transmission

In data sets transmitted as an event from the slave to the master, the time stamp is a double word **after** the actual user data (max. 14 words user data + 2 words time stamps = 16 words). If the data set contains 4 user data words for instance, the time stamp will lie in words 4 and 5.

The time stamp indicates when the event occurred. The time is calculated as a double word in 0.1 ms since midnight. Example: the event was triggered at 14:45.30: the time stamp then has the value:

$$\begin{aligned}
 &14 * 60 * 60 * 1000 * 10 + \\
 &45 * 60 * 1000 * 10 + \\
 &30 * 1000 * 10 = 531300000 \text{ dec.} \\
 &= 1FAAFEAO \text{ hex.}
 \end{aligned}$$

This value is split into one most significant word and one least significant word, each with 16 bits, and stored directly **after** the user data in the data set, with the most significant word first:

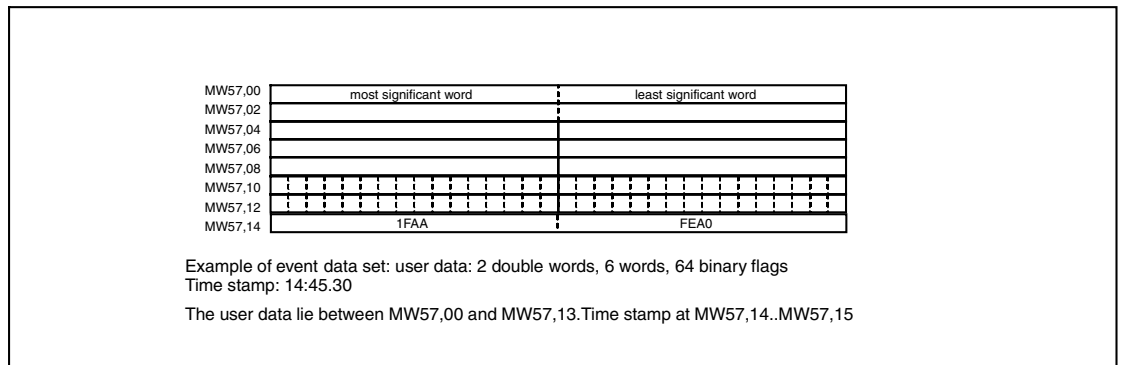


Figure 1-7 Storing event data sets in the double word

1.4.3 Use of the system services

A specific sequence of system services must be observed for starting an RCOM network. This sequence ensures correct initialization of the RCOM protocol.

All system services are triggered by the master with the corresponding connection elements. Nothing needs to be planned in the slaves for the system services. The 07 KP 90 R303, as slave, handles all system services independently.

Important: Please do not confuse the RCOM services Cold start and Warm start with the PLC commands of the same names. The terms “Cold start” and “Warm start” always refer to the RCOM system services in this manual.

Cold start

A Cold start service must be performed after the RCOM master has been initialized. The Cold start can be transmitted either by broadcast to all slaves simultaneously or to each slave individually.

Cold start involves the entire protocol mechanism being reinitialized and the contents of the event queue being cleared. A special cold start event is then triggered in the addressed RCOM slaves. This event is required when operating with ABB MasterPiece systems. It is merely indicated when polling in the case of pure Advant controller networks.

Normalization must always be carried out after a Cold start since, otherwise, it is not possible to transmit data sets.

Warm start

The event queue of one slave (or all slaves) can be cleared with a Warm start service.

The event queue of one slave (or all slaves) can be cleared with a Warm start service. A warm start can be used for resuming communication after transmission errors for instance. This permits master and slave to resynchronize.

Normalization must always be carried out after a Warm start since, otherwise, data sets cannot be transmitted.

Normalization

A slave must be normalized after a Cold start or Warm start. Normalization enables transmission of data sets and events. If a slave is not normalized, it cannot trigger events. The TRANSM connection element then issues a corresponding error message.

If a master polls a non-normalized slave, the POLL connection element signals a corresponding error.

Set clock

When a 07 KP 90 R303 is switched on, its software clock is set to 0:00 hours. You can use the CLOCK connection element to set the clock of the RCOM master and clocks of all slaves to the same time. This is important for evaluating time stamps in the case of event-driven data transmission.

You should set the clock after the Cold start, and clock setting should be repeated if necessary cyclically (e.g. every 24 h).

1.4.4 The RCOM system time

The 07 KP 90 R303 contains a software clock for generating time stamps and for other purposes, and you can also use this software clock in your PLC program. The time information is made available at connection element RCOM at outputs NT, Std, Min and Sek and is updated there approx. every 5 seconds.

The RCOM time starts at 00:00.00 hours when the coupler is switched on.

Connection element CLOCK sets the RCOM clock of the master and sends a Set clock telegram to all slaves (NOD must be set to 255 for this purpose).

Proceed as follows when setting the RCOM time: Read the actual time from the real-time clock and start the CLOCK connection element in the RCOM master with this time (NOD=255 in order to address all slaves). The new time is now transferred to the RCOM clock in the master and in all slaves, and output NT is set to "1" for approx. 5 seconds. If real-time clocks are also to be used in individual slaves, you can use the edge of NT to set these clocks. The master and all slaves now use the same time.

You should use the CLOCK connection element even if the RCOM master does not have a real-time clock (e.g. with time 00:00.00 hours). All time stamps in events are then calculated relative to this arbitrary RCOM time.

1.4.5 Planning procedure

When planning, you should first precisely analyze the required communication relationships in order to avoid subsequent modifications.

You should consider the following questions:

- How many slaves are necessary? Issue the slave numbers (NOD).
- How many data words have to be transmitted for each slave? Define the subdivision of the data into data sets. Issue flag ranges for each data set.
- How must the individual data sets be transmitted? Cyclically? At the request of the PLC program? Event-driven? Define the communication sequences. Do not forget the required system services (e.g. Cold Start and Normalization). Chapter "Examples" shows how such sequences can be implemented.
- How must the PLC program respond to transmission errors? The example program shows one possible solution.

Important planning rules

The following rules must be observed when planning on the 07 KP 90 303:

- IDT (data set ident) may have values between 1 and 255.
- Only an even number of data words may be transmitted in a data set.
- The connection elements may not be skipped once they have been started. Otherwise, this disturbs the logic sequence between the connection element and the 07 KP 90 R303; the connection element may block.
- The connection elements may not be started before the RCOM connection element has run successfully (initializing the coupler). Otherwise, the connection elements may block.
- Data transmission with READ jobs takes approximately twice as long as an event-driven transmission. Consequently, you should give preference to event-driven transmission in the case of time-critical transmissions.
- Max. 14 words are permitted per data set with event-driven transmission. The time stamp lies directly after the user data in two words. The master must know the number of words transmitted if the master wishes to evaluate the time stamp (plan a fixed length).

1.4.6 Hardware handshake

The coupler can handle two types of hardware handshake for communication with a modem: half duplex and full duplex.

The two possible duplex modes are selected on the connection element RCOM with parameter "Dupl".

Hardware handshake relates only to **transmission** of telegrams or jobs. All characters are accepted and interpreted as valid characters at all times in the receive direction.

Please note that a valid CTS signal must be available before transmission of a telegram.

The 07 KP 90 R303 otherwise signals an error. If the modem used does not provide a CTS signal, RTS must be connected to CTS in the cable.

Full duplex

In the case of full duplex mode, the coupler sets its RTS line to "1" after initialization.

Before transmission of characters, it awaits a valid CTS line. The modem may set CTS to zero during transmission in order to stop data flow.

Full duplex should be used for transmission links which provide each transmission direction with its own channel, e.g. modem-zero cables, telephone connections or modem LS-01 of Messrs. Hedin-TeX.

The illustration below shows full-duplex data communication on an 07 KP 90 R303 as RCOM slave.

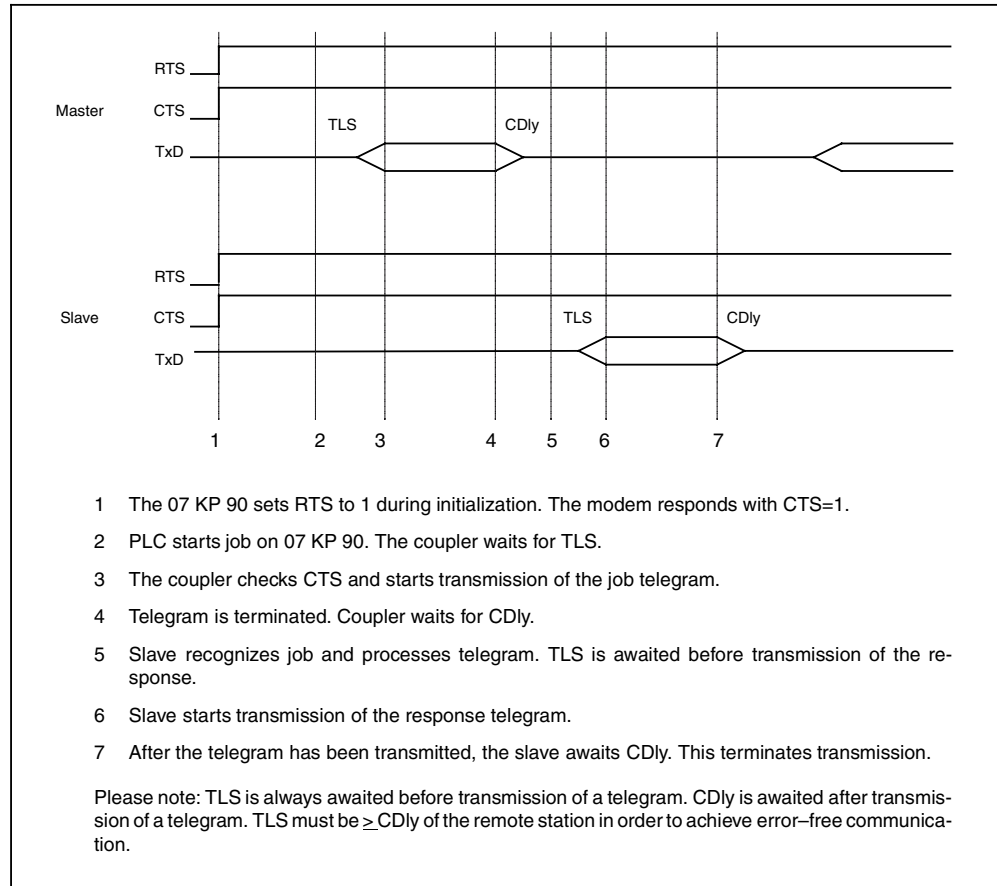


Figure 1-8 Full-duplex data communication

Half duplex

In half duplex mode, the coupler activates RTS before each telegram and then deactivates it again.

Certain modems can thus be switched over from receive mode to transmit mode. This permits a common transmission channel to be used for transmission and reception.

The illustration below is intended to clearly indicate this relationship.

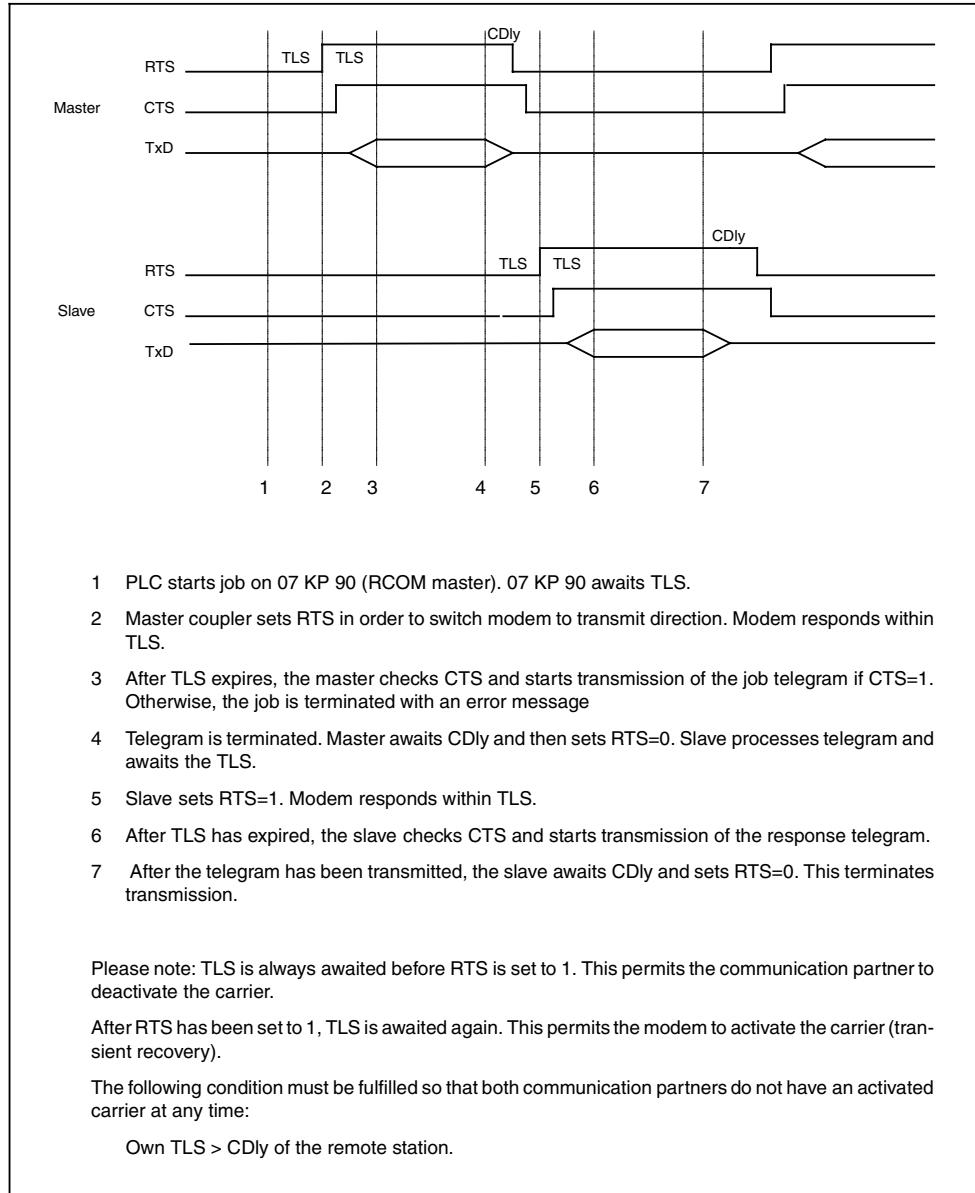


Figure 1-9 Half-duplex data communication

Delay times

Delays which may contribute to increasing transmission reliability can be planned before and after the telegram both in the case of Full duplex mode and in the case of Half duplex mode.

Parameter TLS ("line stab. time") indicates the time which is awaited before transmission of a telegram and after activation of the modem carrier with CTS=1 (in the case of Half duplex only).

Parameter CDLY ("carrier delay") indicates the delay after the telegram.

The following condition must be observed in the case of Full duplex and Half duplex mode in order to guarantee reliable transmission:

Own TLS > CDLY of the remote station.

The two delay times are entered as a number of characters (duration of transmission of a character at the given baud rate) so that longer delays result in the case of lower baud rates.

Since the internal clock runs only with the clock of 10 ms, only multiples of 10 ms are practical. Example: Baud rate = 9600 baud → 1 character = approx. 1 ms, practical time values: 10, 20, 30 etc.

Baud rate = 4800 baud → 1 character = approx. 2 ms, practical time values: 5, 10, 15 etc.

Recommended values for TLS and CDLY are as follows at 1200 baud:

- with half duplex: TLS = 3 characters, CDLY = 2 characters
- with telephone connections: TLS = 2 characters, CDLY = 2 characters
- with full duplex (multidrop or point-to-point): TLS = 2 characters, CDLY = 0 characters.

Please note that these values depend very greatly upon the modem used and should be determined experimentally, in particular in the case of half-duplex links.

1.4.7 Special modems

Integrated RS-485 interface

The RS-485 interface integrated into the 07 KP 90 R303 can be used for connecting several communication modules for max. 500 meters around.

Therefore the following parameters are to be set (example for 1200 and 9600 Baud):

- Baud=9600
Pari=1
Dupl=0
TLS=20
CDly=12
CTO=40
TAT=2000
Mdm=2
NoPr=1

- Baud=1200
Pari=1
Dupl=0
TLS=2
CDly=1

CTO=40
TAT=2000
Mdm=2
NoPr=1

23 WT 90

The 23 WT 90 modem can be operated at the 07 KP 90 R303 with the following parameters:

- Baud=1200
Pari=1
Dupl=0
TLS=20
CDIy=0
CTO=40
TAT=2000
Mdm=3
NoPr=1

Set the switches at the 23 WT 90 modem as follows:

- S1-1: ON
S1-2: ON
S1-3 ON
S2-1: ON
S2-2: OFF
S2-3: ON
S2-4: OFF
S2-5: OFF
S2-6: OFF
S2-7: OFF
S2-8: OFF

Note:

If operating more than one slave with the 23 WT 90, error messages appear on the slaves (error 3002). These error messages have to do with technical features of the 23 WT 90. They don't result in a handicap of communication as long as no error messages appear on the master.

Logem LGH 9600H1

Using this modem for dial-up operation is described in the Chapter 'Dial-up modem'.

1.4.8 Behavior in the case of RUN/STOP and Single Step/Single Cycle

RUN/STOP behavior

After initialization, the 07 KP 90 R303 runs until it is reinitialized. This means that it also accepts jobs if the PLC is stopped.

If you abort a running PLC program, it may happen that you interrupt an RCOM connection element. The 07 KP 90 R303 warns you with a message “command not reset by PLC” or “no PLC reaction” if a connection element does not respond within 2 seconds.

If the coupler is planned as an RCOM slave, it issues the corresponding error message (error message ‘application part not ready’, error number 4020 hex.) to the RCOM master.

If the coupler is planned as the RCOM master, the interrupted job is repeated until the PLC resets the job. This will be the case at the latest the next time initialization is performed by the RCOM connection element.

Single Step and Single Cycle

Since the communication processor monitors the reaction of the PLC in the case of incoming and triggered services, it is not possible to run the RCOM connection elements in Single-step or Single-cycle mode. You should thus always run the communication part of the PLC program in “real time”.

1.5 Using dial-up modems

1.5.1 General

The 07 KP 90 communication processor is capable of handling communication via the telephone network. For this purpose, it can control Hayes-compatible modems (controlled by AT commands).

If you use dial-up modems for data transmission, you must know the following:

- Use only Hayes-compatible modems.
- Deactivate any MNP options which may be available on the modem. In the case of MNP transmission, the temporal relationship between telegrams is lost, and this impedes transmission.

For transmission, a physical connection is necessary, which allows the transmission of breaks and binary characters, without losing the coherence in time (duration of the break signal, intervals of the characters). For most modems this operating mode is called 'direct mode'.

- In the PLC program, the connection must be set up (DIAL connection element) before transmission of RCOM services and it must then be cleared down again (HANGUP).

When commissioning, you should first attempt to address the modem with the operator command "MOD" (refer to Chapter "Operator"). If you enter the following command

```
OPERATOR> MOD AT14 <CR>
```

you should see a table of the most important modem parameters. If this is not the case, the modem will probably have been configured incorrectly (baud rate or parity etc.)

Communication sequence on the RCOM master

You must observe the following sequence in the PLC program of the RCOM master for data transmission:

- Initialize the 07 KP 90 with connection element RCOM. Parameter Mdm=1 (dial-up modems).
- Call the distant station: DIAL. If DIAL is completed and no error is signalled:
- Perform a Cold start or Warm start at the distant station if required. Cold start and Warm start delete the event queue of the called slave. You should **not** use broadcast telegrams with these services but, rather, address the slave explicitly. You must then
- perform normalization. This service **must** always be performed in order to initialize the protocol mechanism for data transmission. Here as well, you should address the slave explicitly with its number and not use broadcast telegrams. Only if normalization does not signal an error, you may:
 - write and read data sets (TRANSM and READ) and
 - poll the slave (POLL and RECV).
- You must then place the telephone back on hook (HANGUP).

In the case of any transmission errors which occur, in particular during normalization, you should place the telephone back on hook with HANGUP and start a new dialing attempt.

Communication sequence on RCOM slaves

No special connection elements need to be planned in the PLC program for regular data transmission (master calls and starts services) in the case of RCOM slaves.

The slave “picks up” the telephone when it rings and then awaits telegrams from the master. If no further telegrams arrive after a waiting time has expired (HANGUP time), the slave “hangs up” automatically.

Event transmission: DIAL in slave

The slave can call the master if it wishes to transmit events to the master.

Only a DIAL connection element needs to be started in the slave for this purpose. Communication runs as follows:

- Slave calls master with DIAL.
- Master answers the telephone.
- After a brief waiting time, the master starts to normalize all slaves planned in the telephone directory. Since only one slave can have called, only the caller will answer correctly.
- The master now **automatically** polls the recognized slave until it signals that the event queue is empty or until the number of polls defined by parameter MaxP is reached. Data sets arriving are transferred to the RECV connection elements in the master.
- The master then “hangs up”.
- The slave also “hangs up” after a waiting time (“Hang-up time”).

Please note:

- The master automatically attempts to poll all slaves when it is called. No POLL connection element is required for this purpose. You use the POLL connection only if the master calls the slave.
- No HANGUP connection element needs to be planned in the slave since the slave does not know when transmission is completed. The slave “hangs up” automatically after expiry of the “hang-up time” (refer to Section “Timeouts”).

1.5.2 Timeout

Whether RCOM telegrams are also actually transmitted with the telephone off hook is strictly monitored both in the master and in the slave.

If no telegrams have been received by the slave or if no services have been started in the master after expiry of the “hang-up time”, the telephone is placed on hook again.

This prevents “wrong callers” who have dialed the wrong number for instance from blocking the telephone continuously.

Since the RCOM slave is not capable of completely monitoring the status of the modem (control **only** with DIAL), you should select a short “hang-up time” for the slave, e.g. 10 seconds.

On the master, the timeout should never respond since the modem can be monitored completely by the PLC program (DIAL and HANGUP). Consequently, you can select a long “hang-up time”, e.g. 30 seconds.

1.5.3 Modem parameters

Correct setting of the modem is very important in order to achieve error-free communication.

Certain parameters can be stored in a non-volatile memory in the modem. All other parameters can be stored on the 07 KP 90 R303 in the Init string of the modem setup. They are then transferred to the modem when the 07 KP 90 R303 is initialized.

The following parameters **must always** be configured so that the 07 KP 90 R303 can control the modem correctly:

- Commands are echoed by the modem
- Acknowledgements from the modem on
- Acknowledgements in plain text
- Break does not clear down connection
- MNP options off
- Data compression off (direct mode)
- Dialing with DTR off
- RTS/CTS handshake between modem and 07 KP 90 R303
- Automatic call acceptance off
- Escape character: '+'

1.5.4 Setup

After initialization, the 07 KP 90 R303 checks whether a dial-up modem is used on the coupler (parameter Mdm=1 on the RCOM connection element). If so, the SETUP file and the telephone directory are read. The 07 KP 90 R303 then expects a Hayes-compatible telephone modem on the RCOM interface.

MODEM_INIT	ATZ^M~~ATI4^M~
DIAL_PREFIX_1	ATDT
DIAL_PREFIX_2	ATDP:
DIAL_PREFIX_3	not used
DIAL_PREFIX_4	not used
DIAL_PREFIX_5	not used
DIAL_PREFIX_6	not used
DIAL_SUFFIX	^M
CONNECT_ANS	CONNECT
MODEM_RING	RING
NO_CARRIER	NO CARRIER
COMMAND	~++++~
MODEM_ANSWER	ATA^M
MODEM_HANGUP	ATH^M~~~~~
MAX_RING_TIME	70
MAX_NO_OF_CALLS	3
CALL_DELAY	2
HANGUP_TIME	30

Figure 1-10 Example of Setup file

The Setup file contains all commands required for controlling the modem. It is pre-configured for the Logem LGH 9600H1 modem and should not be changed if using these modems.

The Setup file is a text file. You can enter it line-by-line via the "CONSOLE" commissioning interface of the 07 KP 90 R303.

The file has a line-by-line structure, i.e. one line defines one parameter.

Caution: Do not forget any of the lines in the file! The keywords at the start of the line may not be changed or omitted.

Two characters in the file have a special significance:

- ~ If you see this character (tilde) in a string, this means that communication to the modem is stopped for one second (waiting time). Certain modems need a pause after certain commands. You should use this character in such cases.
- ^ This character (circumflex) is used to precede a CONTROL character. Thus, for instance: ^M = carriage return.

The parameters are as follows:

1. **MODEM_INIT** is a character string transmitted to the modem upon initialization. Generally, this command resets the modem (e.g. with 'ATZ') and then changes default settings.
2. **DIAL_PREFIX_1..6** are character strings which precede the telephone number. Various prefixes are possible, e.g. in order to permit switchover between dial pulsing and DTMF (Dual Tone Multiple Frequency). The number of the prefix is specified in the telephone directory for the relevant telephone number.
3. **DIAL_SUFFIX** is a character string sent after the telephone number to the modem.
4. **MODEM_RING** is the character string which the modem sends to the 07 KP 90 R303 when the telephone rings.
5. **NO_CARRIER** is the character string which the modem sends if the partner hangs up or if the line has been interrupted.
6. **COMMAND** is the character string which sets the modem to command mode. The character string is transmitted to the modem before the string for hanging up. Certain modems require a brief pause after this. Use "~" in the character string for this. If COMMAND contains no string (entry 'COMMAND'), the 07 KP 90 R303 uses the RTS line as DTR signal for hanging up. This operating mode is described in the Chapter 'Operation with the LGH 9600H1'.
7. **MODEM_ANSWER** is the character string with which the modem accepts an incoming call (off hook).
8. **MODEM_HANGUP** is the character string which causes the modem to hang up. The modem is set to command mode beforehand with the COMMAND character string.
9. **MAX_RING_TIME** is the time in seconds for which the 07 KP 90 R303 waits for a connection to the partner. The time is calculated as of transmission of the dial string.
10. **MAX_NO_OF_CALLS** is the maximum number of attempts made to call the partner. The partner is then recognized as non-reachable (error message in the DIAL connection element).
11. **CALL_DELAY** is the waiting time in seconds between two call attempts.
12. **HANGUP_TIME** is the maximum time which may elapse without services arriving from the PLC or via the RCOM interface. If this time is exceeded, the 07 KP 90 hangs up (refer to section Timeout).

1.5.5 Telephone numbers

The telephone numbers of the individual slaves are also stored in a file. The numbers are read when connection element DIAL is being processed..

The file is a normal text file.

The file has a line-by-line structure, i.e. the telephone number for slave 1 can be found in line 1 etc. Please enter a blank for slave numbers not used. The entry is then deleted.

A line consists of three parts:

- Number of the prefix
- Telephone number
- Any comment

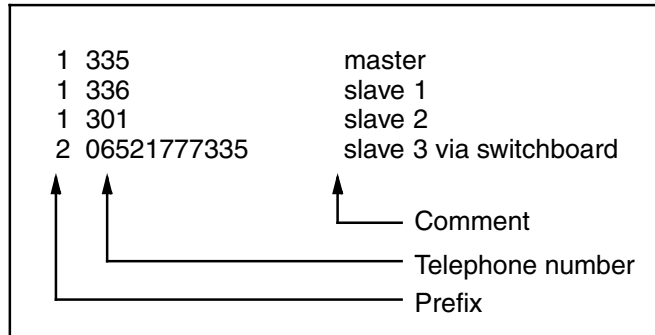


Figure 1-11 Example telephone directory

If a slave is called with connection element DIAL, parameter NOD must be converted to an effective telephone number. The telephone directory is used for this.

The prefix number indicates the number of the DIAL_PREFIX in the Setup file which must be used for this slave. Three parts are transmitted to the modem overall:

- the selected prefix
- the actual telephone number
- the DIAL_SUFFIX from the Setup file.

For example, let us assume that DIAL_PREFIX_1 is "ATDT" and that DIAL_SUFFIX is "^M". When slave 1 is dialed (NOD=1 on DIAL), "ATDT335<return>" is sent to the modem.

The following rules apply to the structure of the telephone directory:

- The file is line-oriented. Each slave corresponds to one line, i.e. parameter NOD on the DIAL connection element corresponds to the line number.
- A line consists of three entries:
- Prefix number
- Telephone number
- Comment
- The entries are separated by blanks (at least one blank).
- The prefix number must be between 1 and 6.
- The telephone number can be any number. **Important:** Do not separate the digits. Use only digits in the telephone number.
- The comment may consist of any characters.

1.5.6 Entering the files

The files are entered as follows via the commissioning interface:

- Enter “PHONE” or “SETUP” on the console, dependent upon the actual file which you wish to change. If you want to change a specific entry, you can also enter “PHONE 3” or “SETUP 5” etc. Editing then starts at the specified line number.
- On the console you will see a line of the file with prefixed line number and a colon. In the next line, you will see “>” beneath the colon. Example (line 4 of the display):
phone 1: 2 06221777335 Slave 1
phone 1> _
- If you want to change the line, simply enter the line again and terminate the entry with <ENTER>. If you do not want to change the line, simply press <ENTER>.
- If you want to delete your entry and enter a line again, press CONTROL-C.
- You can copy parts from the old entry if you have not yet changed the line: in order to do this, press CONTROL-B. A word is copied from the old entry (this is helpful with the SETUP table).
- If you wish to delete an entry in the telephone directory, enter only a blank and then press <ENTER>.
- If you want to quit editing before the last line, enter CONTROL-Z.

The files are stored unchecked in the RAM and remain stored until the next time the coupler is switched off.

If you want to copy the files from the RAM to the EEPROM, enter “SAVE”. Programming may take up to 40 seconds.

If you want to create the files illustrated in this chapter, enter “DEFAULT”. The 07 KP 90 R303 then generates these example files which you can change and then store with SAVE.

Important: Changed Setup data and telephone numbers are lost when you switch off the 07 KP 90 R303 if you have not saved them to the EEPROM beforehand with “SAVE”!

1.5.7 Operation with LGH 9600H1

Special features

In 'direct mode' the LGH 9600H1 (in the following designated as LGH) cannot react on the sequence '+++'. So switching from online to command mode for hang-up is not possible.

For remedy, there is an operating mode of the 07 KP 90 R303, which enables the hang-up by means of the DTR signal (S1/108) of the modem. If the DTR is switched from active to passive, the modem hangs-up immediately and switches to the command mode.

Call-acceptance and making requested connections is only possible with active DTR signal.

Because the 07 KP 90 R303 is not able to make an independent DTR signal available, the RTS signal is used therefore (operating mode 'RTS as DTR'). So the cable for connecting the LGH to the 07 KP 90 R303 is as follows:

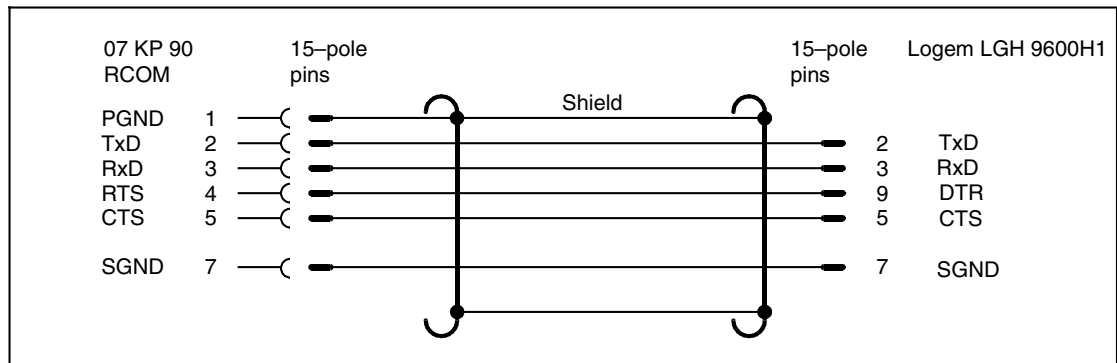


Figure 1-12 Connecting Logem LGH 9600H1 to 07 KP 90 303

Settings at the 07 KP 90 R303

If no character is configured in the setup string 'COMMAND', the operating mode 'RTS as DTR' is used. In that case please pay attention leaving the string MODEM_HANGUP also empty, because for hang-up the DTR signal is used.

This setup is also stored as default setup in the 07 KP 90 R303 and can be called at any time with the command 'DEFAULT'.

```
Operator> default
Operator> show setup
0: MODEM_INIT          ATZ^M~~
1: DIAL_PREFIX_1      ATDTNT
2: DIAL_PREFIX_2      ATDPNT0:
3: DIAL_PREFIX_3      not used
4: DIAL_PREFIX_4      not used
5: DIAL_PREFIX_5      not used
6: DIAL_PREFIX_6      not used
7: DIAL_SUFFIX_       ^M
8: CONNECT_ANS        CONNECT
9: MODEM_RING         RING
10: NO_CARRIER       NO CARRIER
11: COMMAND
12: MODEM_ANSWER      ATA^M
13: MODEM_HANGUP
14: MAX_RING_TIME     70
15: MAX_NO_CALLS     3
16: CALL_DELAY        7
17: HANGUP_TIME       30

OPERATOR>
```

Figure 1-13 Setup 07 KP 90 R303

Settings at the LGH 9600H1

When starting, the LGH is to be set to the following parameters permanently, e.g. with a terminal connected directly to the LGH:

```
at&v
Version 2.02 D
F2 E1 L1 M1 Q0 X4 V1 P \Q2 \G0 \A1 \C0 \L0 \N1 \X1 \K0 \B3 %C1 %E1 %M0 &L0 &I0
&Y0 &X0 &G0 &M0 &C2 &D2 &R1 &S1
S00=000 S01=000 S02=043 S03=013 S04=010 S05=008 S07=100 S08=002 S10=002 S12=050
S20=255 S26=004 S28=000 S37=001 S39=017 S40=019 S45=000 S46=060 S50=002 S51=004
S60=000 S61=000 S80=000 S81=000 S100=042 S101=000 S102=000

OK
```

Figure 1-14 Settings LGH 9600H1

During the input of settings make sure, that after the input of the parameter 'ATF2' the LGH is set permanently to 1200 Baud and that the terminal also is to be set to 1200 Baud.

Store these parameters by means of 'AT&W' in the non-volatile memory of the LGH.

The DIL switches at the LGH are to be set that way, that the basic setting 0 is set in the software mode (all switches S5.1 to S5.5 at the front panel of the unit are set to 'OFF').

1.6 Operator

1.6.1 What is "Operator"?

The second serial interface "CONSOLE" of the 07 KP 90 R303 is also referred to as the operator interface below. The user can enter commands for the communication module at this interface, and the coupler can issue signals and messages via the interface.

There are two main applications for this interface: generation of the telephone directory and tracing the communication sequence during commissioning.

In order to use the CONSOLE interface, you will need a simple terminal or a PC with terminal emulation. The terminal must be set to 9600 baud, 8 data bits, no parity and one stop bit.

1.6.2 Fault-finding: Debug

The user can display all telegrams and all services processed by a 07 KP 90 R303 on the operator console.

There are two levels: In the first level, a message containing the most important parameters is output for each telegram received or transmitted by a slave and for each service started by the PLC.

In the second level, the coupler issues a message with each important action (including internal operations). Thus, received data words and transmitted telegrams are displayed for instance, as well all status changes in the RCSW (rcom status word) and all status changes in the event queue, etc. The second debug level will probably be of interest in only a few cases. The first level suffices for testing (debugging) the PLC program.

The two levels can be set with input "Dbg" on connection element "RCOM" (0: no debug messages, 1: debug level 1, 2: level 2). After initialization, the debug level can be set to "1" by entering command

```
OPERATOR> debug 1
```

(corresponding to level 2). Deactivate the messages with

```
OPERATOR> debug 0
```

After commissioning, you should deactivate the messages for normal operation of the coupler (parameter Dbg = 0).

All operator messages have the following appearance:

- typ-I-identification text
- typ-W- identification text
- typ-E- identification text
- typ-F- identification text

Where:

- typ: Three letters indicating the origin of the message, i.e. "MST" for services performed by a master, "TEL" for telegrams received by a slave, "RPL" for responses transmitted by a slave, etc.
- I/W/E/F provides information on the type of message:
 - 'I' is information serving to trace the sequence (debug levels 1 and 2).
 - 'W' is a warning which occurs if telegrams for which no connection element is planned arrive for instance. Warnings do not disturb the sequence in the coupler but do indicate a planning error.
 - 'E' 'E' is an error message of the coupler, i.e. if an addressed slave does not respond. Errors disturb the coupler when processing the current service but can frequently be remedied by repeating the service. LED ERR lights in the event of an error. If it has been possible to remedy the error, the LED goes out again.
 - 'F' is a fatal error which cannot be remedied. Communication via RCOM and to the PLC is aborted by the coupler and only operator entries are then possible. LED ERR lights steadily and the RUN LED goes off. The coupler can be reactivated only by Reset (key or power off/on).
- "identification" is an abbreviation of the error message.
- "text" contains the actual message in plain text.

1.6.3 Operator commands

The following operator commands can be entered on the console:

HELP

Outputs a help text on the available commands.

TIME

Outputs the current RCOM system time.

EVENT

Outputs a table with the events present in the event queue. The table contains an event number, the event type (40: event with data set, 41: Cold start event), data set number and length and the time when the event occurred.

RCOM

Outputs the table with the current RCOM parameters. IMPORTANT: All times are specified in milliseconds. Some of the times are increased slightly by the coupler.

RCSW

Outputs the RCOM status word (RCSW).

SHOW SETUP

Outputs the setup data. Refer to section "Setups for dial-up modems".

SHOW PHONE

Outputs the telephone directory. Refer to section "Setups for dial-up modems".

PHONE <n>

Editing the telephone directory as of entry "n". If you omit "n", editing is carried out as of the first entry. Refer to section "Setups for dial-up modems".

SETUP <n>

Editing the setup data as of entry "n". If you omit "n", editing is carried out as of the first entry. Refer to section "Setups for dial-up modems".

SAVE

Saving the telephone setup and data on EEPROM. Refer to section “Setups for dial-up modems”.

HANGUP

Hangs up the telephone (only if a dial-up modem is connected).

DIAL n

Dials the telephone number stored under entry “n” in the telephone directory (only if a dial-up modem is connected).

MOD string

Sends “string” to the modem (only if a dial-up modem is connected). The responses of the modem are displayed. Example: “MOD AT14 <ENTER>” → the modem responds with a table of the most important parameters.

DEBUG <n>

Sets the debug level to “n”. This remains valid until the next initialization by the PLC. If you omit “n”, the current debug level is displayed.

1.7 Error codes

Various errors identified with numbers may occur when using the RCOM coupler. The error number is indicated at output FeNR of the corresponding connection elements.

The errors can be subdivided into two groups: recoverable errors and fatal errors.

Recoverable errors

The recoverable errors include all errors occurring during transmission of telegrams (character losses, parity errors etc.). In some cases, the coupler attempts retransmission in the case of transmission errors; the number of attempts can be specified on the RCOM connection element (parameter Retr).

In the case of recoverable errors, LED "ERR" lights briefly. The LED goes out again as soon as the error has been remedied.

Fatal errors

If fatal errors occur, the coupler terminates both communication with the AC31 CPU and communication with the RCOM partners.

It is still possible to make entries via the operator interface. A Reset is required in order to reactivate the coupler, e.g. by switching off and switching back on or by using the Reset key. If you use the Reset key, you must abort the PLC program, and restart it in order to reinitialize the 07 KP 90.

LED "ERR" lights steadily in the case of fatal errors. In addition, LED "RUN" goes out, thus indicating that no further communication is occurring.

Table of error codes

The following table contains all possible error codes (hexadecimal and decimal), their cause and whether the error is fatal or not.

Error code		Cause	Re- marks
(hex)	(dec.)		
0000	0	No error	
0001	1	PLC does not respond (possible cause: missing connection element or PLC stopped)	
1001	4097	Event queue full	
2001	8193	Initialization error: Incorrect address (parameter NOD)	
2002	8194	Initialization error: Incorrect baud rate (parameter Baud)	
2005	8197	Initialization error: Incorrect parity (parameter Pari)	
2006	8198	Initialization error: Incorrect duplex mode (parameter Dupl)	
2007	8199	Initialization error: Incorrect line stab. time (parameter TLS)	
2008	8200	Initialization error: Incorrect carrier delay (parameter CDly)	
2009	8201	Initialization error: Incorrect char. timeout (parameter CTO)	
2010	8208	Initialization error: Incorrect turnaround time (parameter TAT)	
2011	8209	Initialization error: Incorrect retransmissions (parameter Retr)	
2012	8210	Initialization error: Incorrect number of preambles (parameter NoPr)	
2015	8213	Initialization error: Incorrect modem type (parameter Mdm)	
2018	8216	Initialization error: Incorrect debug level (parameter Dbg)	
3001	12289	No CTS during transmission	
3002	12290	Timeout, no response telegram	
3003	12291	Telegram error (incorrect length code)	
3004	12292	Telegram error (incorrect checksum)	
3005	12293	Incorrect slave responding	
3006	12294	Telegram error (incorrect response code)	
4001	16385	Service not known	
4002	16386	Incorrect length entry (parameter LEN)	
4003	16387	Incorrect slave number (parameter NOD)	
4004	16388	Incorrect network number (parameter NET)	
4005	16389	Incorrect data set number (parameter IDT)	
4006	16390	Response telegram does not match service	
4007	16391	Slave not normalized	
4008	16392	Event queue blocked (not normalized)	
4010	16400	Invalid time (connection element CLOCK)	
4020	16416	Slave responds: "application part not ready" (e.g. FREI = 0)	
4030	16432	Slave responds "illegal command"	
5000	20480	Coupler not initialized (after Reset)	1)
6001	24577	Telephone directory or setup data not found	
6002	24578	Incorrect entry in telephone directory or setup data	
6003	24579	No telephone modem planned	
6004	24580	Modem not yet online (service started without DIAL)	
6005	24581	Modem already online (repeated DIAL without HANGUP)	
7000	28672	EEPROM cannot be programmed	
7001	28673	EEPROM cannot be erased	
FFFF	-1	Internal error	fatal
Remarks:		1) Requires restart of the PLC (abort program and restart of the program)	
		fatal Fatal error	

Table 1-1 Table of error codes

1.8 Operator messages

The tables on the next few pages contain all possible messages which may be output on the operator interface and their significance.

Certain messages contain designations for services which are specified in the following table.

Abbreviated	Name of the service	Triggered in the master with CONNECTION ELEMENT	Handled in the slave with CONNECTION ELEMENT
Norm comm part	Normalize communication part	1)	3)
Quer comm part	status query communication part	1)	3)
Cold start	cold start	COLDST	3)
Warm start 1	block all blocks	WARMST	3)
Warm start 2	block unique blocks	1)	3)
set clock	set clock	CLOCK	3)
Norm user part	Normalize user part	1)	3)
Norm all blocks	Normalize all blocks	NORMAL	3)
Norm sep blocks	Normalize separate blocks	1)	3)
Write dataset	write dataset	TRANSM	RECV
Write cntrl	write data to control register	1)	3)
Read dataset	read dataset	READ	READ_S
Event request	event request	POLL	3)
Repeat read	repeat read command	2)	3)
Repeat write	repeat write command	2)	3)
Dial	dial up slave	DIAL	3)
Hangup	hangup phone	HANGUP	3)

Remarks:

- 1) not used as master on 07 KP 90 R202
- 2) is performed automatically in the case of transmission errors
- 3) is handled automatically in the 07 KP 90 R202

Table 1-2 Identification of services

In the table showing all messages, the first column specifies the actual message of the communication module. The next column specifies the significance of this message and the third column shows remarks which may indicate the cause of the error.

Table 1-3 Significance of the message

Message	Description	Significance	Rmk.
-EPL-E-PLCTO	service timeout, no PLC reaction	An event which has arrived has not been fetched by the PLC. RECV connection element missing	Remark 1
-EPL-I-EVENT	event – ds: ..., len: ..., ..	Event arrived	
-EPL-I-PLCACC	service accepted by PLC	Service accepted by PLC	
-EPL-I-SYSMES	system message	System event has arrived (is ignored)	
-EPL-W-PLCREJ	service rejected by PLC,	Event rejected by PLC (FREI in the case of RECV = 0)	
-ERR-F-FATAL	fatal error, communication canceled	Fatal error has occurred, communication has been terminated	
-EVT-I-BLOCK	Blocking event queue	Blocking event queue	
-EVT-I-CLEAR	Clearing event queue	Clearing event queue	
-EVT-I-DEBLK	Deblocking event queue	Enable event queue	
-EVT-I-GET	event queue empty	Event queue is empty	
-EVT-I-GET	getting event	Fetching event from queue	
-EVT-I-PUT	putting event	Inserting event in queue	
-EVT-W-PUT	event-queue full	Event queue is full	
-INI-E-COMGRP	error reading com group, code	Error reading special flags for communication processors	Remark 2
-INI-E-EXTINI	External init error, ...	Error during initialization	2
-INI-E-GRESI	error resetting PLC communication, ...	It has not been possible to reset system bus communication	2
-INI-E-MOD	error initializing telephone modem, ...	It has not been possible to initialize telephone modem	3
-INI-E-OCCUP	error occupying PLC, ...	It has not been possible to assign PLC	2
-INI-E-OPER	error initializing operator, ...	It has not been possible to initialize commissioning interface	3
-INI-E-PARAM	error in parameter, ...	Error in parameter	1
-INI-E-RCOM	error during RCOM-init, ...	It has not been possible to initialize RCOM mechanism	
-INI-E-RCOM	error initializing RCOM-channel, ...	It has not been possible to initialize RCOM interface	3
-INI-E-READ	error reading parameters, ...	It has not been possible to read RCOM parameters	2
-INI-E-RS	error reading RUN/STOP, ...	It has not been possible to read RUN/STOP switch	2
-INI-E-RSINI	error initializing PLC communication, ...	It has not been possible to initialize system bus communication	2
-INI-I-CHECK	Checking RCOM-parameters	Checking RCOM parameters	
-INI-I-COMGRP	waiting for valid com group	Waiting for valid communication area (in special flag for communication processors)	
-INI-I-EXTINI	External init done, start flag: MW nn,0	External initialization completed, communication flag range = MW ...	
-INI-I-EXTINI	waiting for external init	Waiting for external initialization	
-INI-I-GRES	Resetting PLC communication	Resetting system bus communication	
-INI-I-RUN	waiting for RUN-switch	Waiting for RUN/STOP switch = RUN	
-KPM-I-EXIT	exit main loop, reason code ...	Quitting RCOM communication; cause ...	Remark 4
-KPM-I-GOODM	good morning!!	It is midnight	
-KPM-I-LCNT	Lifecount ...	Cycle counter set to ...	
-MOD-E-DIAL	Cannot connect	It has not been possible to establish dial-up connection	5

Message	Description	Significance	Rmk.
-MOD-E-DIAL	Modem already connected	Modem is already on line	5
-MOD-E-DIAL	No modem available (modem type = 0)	No modem planned	1
-MOD-E-ENTRY	Bad entry in phone file	Entry error in telephone directory	5
-MOD-E-HANGUP	Cannot hangup	It has not been possible to hang up	3
-MOD-E-INIT	Error during modem init	Error during modem initialization	3
-MOD-E-NOFILE	No valid files on EEPROM	No valid setup/telephone files on EEPROM	5
-MOD-E-RING	No modem available (modem type = 0)	No modem planned	1
-MOD-I-ANS	Answer ...	Modem response: ...	
-MOD-I-ANSCMP	Compare ... - ...	Comparing modem response with ...	
-MOD-I-ANSCMP	Strings are equal	Strings are identical	
-MOD-I-DIAL	Connected	It has been possible to set up connection (dial)	
-MOD-I-DIAL	Dialing node ...	Dialing station ...	
-MOD-I-DIAL	Ring (...)	Dialing station ...	
-MOD-I-HANGUP	Hangup phone	Hanging up	
-MOD-I-INIT	Answer ...	Modem response:...	
-MOD-I-INIT	Init modem (...)	Initializing modem	
-MOD-I-RING	Connected	It has been possible to set up connection (going off hook)	
-MOD-I-RING	Ring received	Telephone ringing	
-MOD-W-DIAL	Retry ...	Retry dialing	
-MOD-W-RING	Cannot connect	It has not been possible to set up connection (going off hook)	Remark 5
-MST-E-ADDR	Error reading reply (address), ...	It has not been possible to read address from response	6
-MST-E-DATA	Error reading reply (data), ...	It has not been possible to read address from response	6
-MST-E-HEADR	Error reading reply (header), ...	It has not been possible to read header from response	6
-MST-E-LCODE	Illegal length-code in reply	Incorrect length code in response	6
-MST-E-POSTA	Error reading reply (checksum/postambles),...	It has not been possible to read checksum/trailer from response	6
-MST-E-PREA	Error reading reply (preambles), ...	It has not been possible to read leader from response	6
-MST-E-RES	Command not reset by PLC	Command not acknowledged by PLC	7
-MST-E-SEND	Error sending telegram, ...	It has not been possible to transmit job	3
-MST-E-SUM	Checksum error in reply	Checksum error in response	6
-MST-I-POLL	Checking slave %3d	Checking whether slave ... has dialed	
-MST-I-POLL	Polled slave %3d, result ...	It has been possible to poll slave ...; result ...	
-MST-I-RESULT	..., result ...	Service ... terminated; result ...	
-MST-I-SERV	...	Service ... started	
-MST-W-NOSRV	No services within hangup time	No service within the hangup time, now hanging up	Remark 1
-MST-W-RETRY	Retry ...	Retrying job telegram	6
-OPR-E-CMD	Unknown command '...'	Unknown operator command	
-OPR-E-OCCUP	Error occupying PLC, ...	It has not been possible to assign PLC	2
-OPR-E-RELEA	Error releasing PLC, ...	It has not been possible to release PLC	2
-OPR-I-INIT	Operator init done	Initialization commissioning interface terminated	
-PLC-E-GETEND	error reading line area from plc, ...	Error in system bus communication	2
-PLC-E-GETEND	error reading rx area from plc, ...	Ditto	

Message	Description	Significance	Rmk.
-PLC-E-GETEND	error reading tx area from plc, ...	Ditto	
-PLC-E-GETKP	error reading kp number from plc	Ditto	
-PLC-E-GETREQ	error reading com group from plc, ...	Ditto	
-PLC-E-GETREQ	error reading line area from plc, ...	Ditto	
-PLC-E-GETREQ	error reading rx area from plc, ...	Ditto	
-PLC-E-GETREQ	error reading tx area from plc, ...	Ditto	
-PLC-E-SETEND	error writing control block to plc, ...	Ditto	
-PLC-E-SETEND	error writing rx area to plc, ...	Ditto	
-PLC-E-SETEND	error writing tx area to plc, ...	Ditto	
-PLC-E-SETREQ	error writing control block to plc, ...	Ditto	
-PLC-E-SETREQ	error writing rx area to plc, ...	Ditto	
-PLC-E-SETREQ	error writing tx area to plc, ...	Ditto	
-PLC-W-GETEND	Timeout during read com group from plc, abort: ...	Timeout during system bus communication, error during abort: ...	Re-mark 2
-PLC-W-GETEND	Timeout during read line area from plc, abort: ...	Ditto	
-PLC-W-GETEND	Timeout during read rx area from plc, abort: ...	Ditto	
-PLC-W-GETEND	Timeout during read tx area from plc, abort: ...	Ditto	
-PLC-W-SETEND	Timeout during write control block to plc, abort: ...	Ditto	
-PLC-W-SETEND	Timeout during write rx area to plc, abort: ...	Ditto	
-PLC-W-SETEND	Timeout during write tx area to plc, abort: ...	Ditto	
-RCS-I-SET RCSW	set to ...	RCOM status word set to ...	
-RDS-E-PLCTO	service timeout, no PLC reaction	PLC not responding to read job (READ_S connection element missing)	Anm.1
-RDS-I-PLCACC	service accepted by PLC	Service "read data set" accepted by PLC	
-RDS-W-PLCREJ	service rejected by PLC, ...	Service "read data set" rejected by PLC (FREI with READ_S=0)	
-RPL-E-REPLY	internal error: ...	Internal error when setting up response	Rmk. 7
-RPL-E-REPLY	reply error: ...	Error in response telegram	5,6
-RPL-I-LEN	... data bytes in reply	... data bytes in response	
-RPL-I-REPLY	...	Response: ...	
-RPL-I-REPLY	..., result ...	Error ... in response	Rmk.6
-SCL-I-TIME	date: %ld, time %ld	New RCOM time ... arrived	
-SLV-E-ADDR	error reading telegram (address), ...	It has not been possible to read address from job	Rmk.6
-SLV-E-BREAK	error checking for break	It has not been possible to check BREAK	3
-SLV-E-DATA	error reading telegram (data), ...	It has not been possible to read data from job	6
-SLV-E-HEADR	error reading telegram (header), ...	It has not been possible to read header from job	6
-SLV-E-LCODE	illegal length-code in telegram	Incorrect length code in job	6
-SLV-E-POSTA	Error reading telegram (checksum/postambles), ...	Incorrect checksum/trailer in job	6
-SLV-E-PREA	error reading telegram (preambles), ...	It has not been possible to read leader from job	6
-SLV-E-RES	Command not reset by PLC	Command has not been cancelled by PLC	7
-SLV-E-SEND	Error sending reply, ...	It has not been possible to send response	3

Message	Description	Significance	Rmk.
-SLV-E-SUM	Checksum error	Checksum error	6
-SLV-I-ADR	Not my address (...)	Job not for me but for slave ...	
-SLV-I-NOREP	No reply sent (broadcast request)	No response transmitted (broadcast job)	
-SLV-I-RESULT	..., result ...	Service terminated; result ...	
-SLV-I-SERV	...	Service detected	
-SLV-W-MODE	Event queue blocked	Event queue barred (normalization missing)	Re-mark 1
-SLV-W-MODE	Slave mode program	Data transmission barred (normalization missing)	1
-SLV-W-NOSRV	No services within hang-up time	No jobs have arrived within the hang-up time, now hanging up	
-TEL-E-SERV	Internal error: ...	Internal error	8
-TEL-I-LEN	... data bytes in request	... data bytes in job	
-TEL-I-SERV	...	Service ... detected	
-TEL-I-SERV	... Ds: ..., Len: ...	Service ... detected; data set ...; length ...	
-WDS-E-PLCTO	Service timeout, no PLC reaction	PLC not responding to write job (RECV connection element missing)	Re-mark 1
-WDS-I-PLCACC	Service accepted by PLC	Service "Write data set" accepted by PLC	
-WDS-W-PLCREJ	Service rejected by PLC, ...	Service "Write data set" rejected by PLC (FREI with RECV = 0)	

Remarks:

- 1 A planning error has probably occurred. Check whether all required CEs are present and whether the correct parameters have been assigned to them.
- 2 Error during system bus communication. It is either remedied automatically or it leads to fatal errors. May be triggered by 907 AC 1131, e.g. when transmitting programs.
- 3 There is probably a fault in the cable. Check the wiring of RTS and CTS.
- 4 The following causes are possible:
 - 2: Reinitialization has been performed by the PLC (RCOM connection element started)
 - 3: RUN/STOP switch has been set to STOP
 - 4: A fatal error has occurred
- 5 There is probably an error in modem control. Check: Modem settings, SETUP and PHONE data.
- 6 A transmission error has occurred. Check all RCOM parameters and timeout times.
- 7 This occurs if the CPU is set to STOP.
- 8 Internal error. This may not occur. Attempt to reset the unit and reinitialize it.

1.9 Instructions for RCOM communication commissioning

Coupler settings for the example ‚Wählmaster_s90‘ and ‚Wählslave_s90‘ with 907 AC 1131.

Prerequisite: The modems MUST have the feature for direct mode!!

Hint: It is recommended to provide several PCs for checking the data traffic via the Console interfaces of the couplers 07 KP 90. Terminal programs are required (PCPLUS or Hyper Terminal; settings: 9600 baud, 8 data bits, 1 stop bit, parity none).

1.9.1 Presetting of the dial-up modems: e. g. ELSA Microlink 14.4TQ

The modem is connected with the delivered modem cable to the PC (1:1, no cross-connections, because modems are actually extension cables only!)

Modem 9-pole male	Adaptor	PC 9-pole female
DCD	1	1 CD
RxD	2	2 RxD
TxD	3	3 TxD
DTR	4	4 DTR
SGND	5	5 SGND
DSR	6	6 DSR
RTS	7	7 RTS
CTS	8	8 CTS
RI	9	9

Figure 1-15 Modem cable

The following setting is performed by means of a terminal program:

AT&F	Loads standard configuration
ATX0	Ignores dial-up signal / busy signal (important in case of internal connections only, in case of external connections not faulty)
ATE1	The modem echoes the commands.
ATN1	Switches modems to direct mode – attention, no all modems are able to do so!!
AT&W	Non-volatile storing of the extended configuration profile

The settings can also be realized via the SETUP of the coupler, e.g.

```
OPERATOR> show setup
0: MODEM_INIT      AT&F^M~~ATZ^M~~ATX0^M~~ATE1^M~~AT\N1^M~~
1: DIAL_PREFIX_1  ATDT
2: DIAL_PREFIX_2  ATDP
etc.
```

We recommend setting of the modems directly via a terminal program (as described above).

1.9.2 Connecting modems with the RCOM interface of the 07 KP 90

Direct connection between 07 KP 90 and the modem is established with the following modem cable (1:1, no cross-connections, because modems are actually extension cables only!)

Hence RxD-RxD / TxD-TxD / RTS-RTS / CTS-CTS

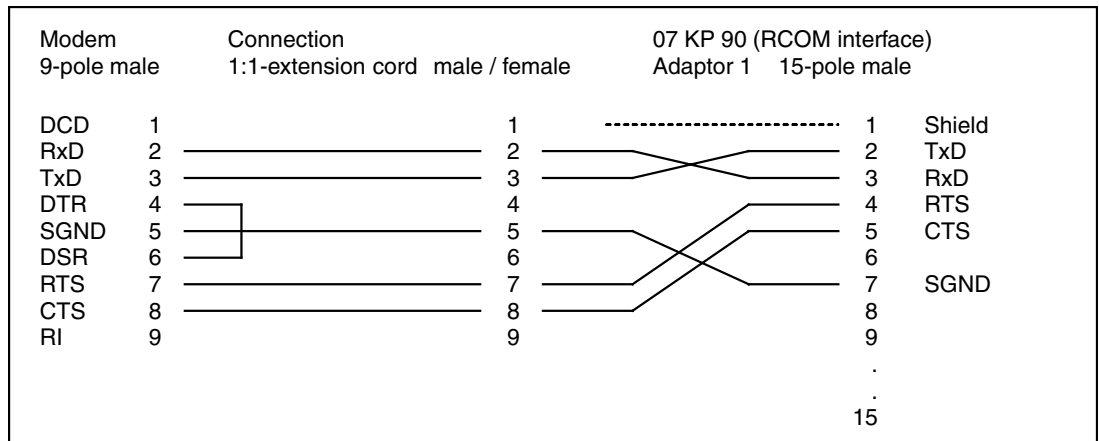


Figure 1-16 First variant of interface cable between modem and 07 KP 90

or also:

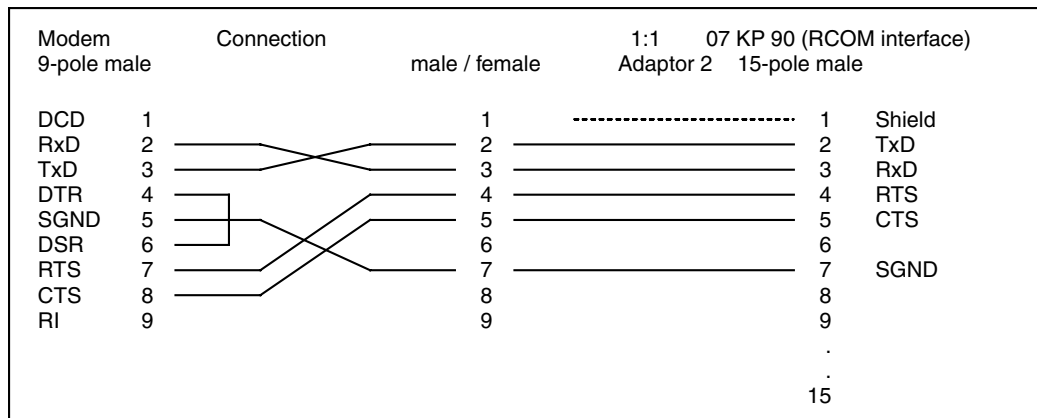


Figure 1-17 Second variant of interface cable between modem and 07 KP 90

1.9.3 Connecting terminal-PC with the Console interface of 07 KP 90

You can use e.g. Hyper terminal or PCPLUS as terminal program.

Connect adapter between 07 SK 90 (AC31 programming cable – to PC) and Console interface of 07 KP 90 (to view the parameters and the data traffic with the PC).

In this case RxD/TxD and RTS/CTS are cross connected!

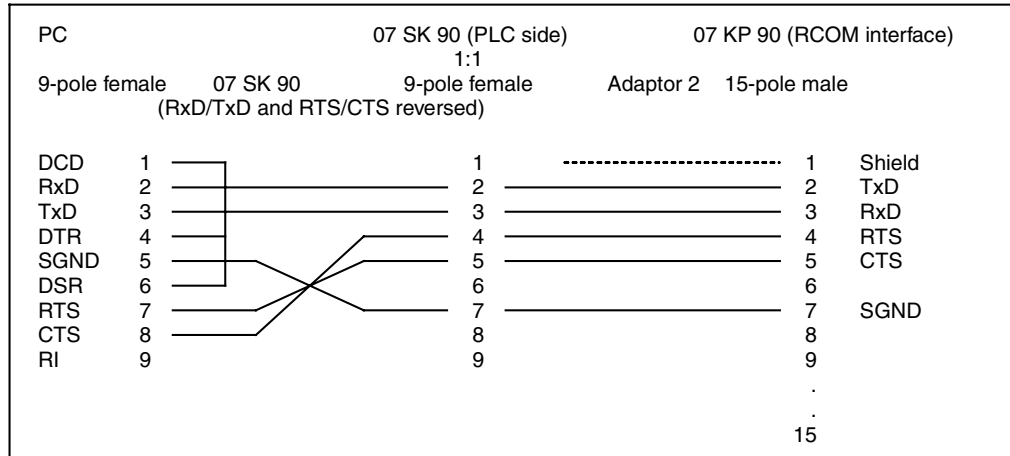


Figure 1-18 First terminal cable

1.9.4 Parameterization of coupler 07 KP 90

The parameterization is done via the Console interface of the coupler. A coupler can execute commands, in case an according command is entered following the „OPERATOR>“. E.g. „HELP“ lists all available commands.

```
OPERATOR> help
```

```
available commands:
```

```
help.....this text
time.....show RCOM's system time
event.....show event-queue
rcom.....show RCOM parameters           Data at the RCOM
                                           connection element

rcsw.....show RCOM status word
show setup.....show setup-file
show phone.....show telephone directory
phone <slave>.....modify telephone directory
setup <entry>.....modify setup entries
save.....save setup/phone
default.....create default setup/phone
hangup.....hangup phone
dial <slave>.....dial a slave
mod <command>.....send a command to modem
debug <level>.....show/set debug level
```

Setup for the MASTER:

```
OPERATOR> show setup
 0: MODEM_INIT      ATZ^M~~      Sets modem to default values
 1: DIAL_PREFIX_1  ATDT          DTMF (Dual Tone Multiple Frequency)
 2: DIAL_PREFIX_2  ATDP          Pulsing
 3: DIAL_PREFIX_3  not used
 4: DIAL_PREFIX_4  not used
 5: DIAL_PREFIX_5  not used
 6: DIAL_PREFIX_6  not used
 7: DIAL_SUFFIX    ^M            CR (carriage return), following the number
 8: CONNECT_ANS   CONNECT       Modem response, if connection is established
 9: MODEM_RING     RING          Modem response, if call is detected
10: NO_CARRIER   NO CARRIER  Modem response, in case of line OFF
11: COMMAND       ~~~+++~~~    Switches modem to command mode
12: MODEM_ANSWER  ATA^M        Modem response, if pick-up the receiver is performed
13: MODEM_HANGUP  ATH^M        Modem response, if hang-up the receiver is performed
14: MAX_RING_TIME 70           sec., the 07KP90 waits for connection
15: MAX_NO_CALLS  3           max. dial-up attempts as of Dial, before error occurs
16: CALL_DELAY    7           sec. between two dial-up attempts
17: HANGUP_TIME   30          sec., after hang-up is performed in case of silence
```

Setup for the slave 1:

```
OPERATOR> show setup
 0: MODEM_INIT      ATZ^M~~
 1: DIAL_PREFIX_1  ATDT
 2: DIAL_PREFIX_2  ATDP
 3: DIAL_PREFIX_3  not used
 4: DIAL_PREFIX_4  not used
 5: DIAL_PREFIX_5  not used
 6: DIAL_PREFIX_6  not used
 7: DIAL_SUFFIX    ^M
 8: CONNECT_ANS   CONNECT
 9: MODEM_RING     RING
10: NO_CARRIER   NO CARRIER
11: COMMAND       ~~~+++~~~
12: MODEM_ANSWER  ATA^M
13: MODEM_HANGUP  ATH^M
14: MAX_RING_TIME 70
15: MAX_NO_CALLS  3
16: CALL_DELAY    7
17: HANGUP_TIME   5
```

Setting the Setup parameters:

In case of setting the Setup parameters always the complete line is to be input in the line editor. E.g. MODEM_INIT.....ATZ^M~~<CR>

```
OPERATOR> setup
                6 blanks!
                .....
setup  0: MODEM_INIT      ATZ^M~~
setup  0>

setup  1: DIAL_PREFIX_1  ATDT
setup  1>

setup  2: DIAL_PREFIX_2  ATDP
setup  2>
```

Creating telephone directory (Master and slave x identical):

The input of the telephone numbers is performed in the line editor. Always the complete line is to be input. E.g. MODEM_INIT.....ATZ^M~~<CR>

In telephone installations mostly a preceding 0 is to dial-up, hence
0W03091772102

In the example internal telephone numbers are used.

```
OPERATOR> show phone
0: 1 2102          Master          Entry 0 is always the master
1: 1 2153          Slave 1          Slave 1
2: 1 2155          Slave 2          Slave 2 etc
* -----
* ----- Consecutive number, Node at the DIAL, master is
           always 0
* ----- Prefix, which is output during dial-up (ATDT is
           according to setup the prefix 1, ATDP is
           prefix 2 - actually there is only prefix 1
           existing)
           Hence here e.g. ATDT2102
```

Saving settings:

The settings are saved by means of the command „SAVE“.

```
OPERATOR> save
programming eeprom, please wait...
programming done.
OPERATOR>
```


Viewing the settings of the RCOM connection element in the coupler:

```
OPERATOR> rcom
RCOM parameters:
    net:      0          always 0
    address:  1          0=Master, 1=slave1, 2=slave2 ...etc....
    bitrate:  9600
    parity:   0
    duplex:   1
    line st. time: 3
    carrier delay: 2
    char. timeout: 52
    turnaround t.: 6000
    retransmiss. : 3
    max no. poll: 20
    type of modem: 1
    preambles: 3
    plc timeout: 2000
    rcom typ:  1          0=RCOM; 1=RCOM+
    digi time: 0
```

General hints:

Direct modem commands can be input via the Console interface also, if one precedes the command „MOD“. E.g. a direct command as „ATDT 0W91772102“ is input as

```
MOD ATDT 0W91772102
```

Otherwise the commands listed under „HELP“ can be executed, e.g.

```
OPERATOR>DIAL 1          ;dials-up slave 1
```

The debug level also can be changed. As only one processor is present, this influences the system time.

```
OPERATOR> debug 0          no outputs
OPERATOR> debug 1          most important outputs
OPERATOR> debug 2          all outputs
```

Debug 1 is normal. Debug 2 should be used **only** for IBN, because thereby the system time of the coupler is prolonged.

2.1 Features of the module 07 KP 93 R1163

The communication module 07 KP 93 R1163 is an interface module with 2 serial MODBUS RTU interfaces.

Devices which use the MODBUS RTU protocol for the communication can be coupled to the Advant Controller 31 system using this communication module.

The most essential features of the communication module are:

- 2 serial interfaces, can be used as EIA RS-232 or EIA RS-485 (COM3, COM4) as desired.

Possible operating modes:

COM3 COM4

Master Slave (Master-Master is not possible)

Slave Master

Slave Slave

- The communication to AC31 basic units is performed via function blocks which are part of the programming software 907 AC 1131.

2.2 Communication via the MODBUS interfaces COM3 and COM4

The communication between the basic unit 07 KT 9x and the coupler 07 KP 93 is performed via the networking interface using function blocks of the programming software 907 AC 1131 (see section List of function blocks).

The software 907 KP 93 is not required to perform the programming of the basic units 07 KT 95, 07 KT 96, 07 KT 97, and 07 KT 98. The required function blocks are part of the ABB libraries in the programming software 907 AC 1131.

2.3 List of function blocks for 07 KP 93

The function blocks are part of the programming software 907 AC 1131.

Library: ABB-BIB4.LIB or COM_S90_V41.LIB

The communication between the basic units 07 KT 9x and 07 KP 93 is performed using the following function blocks:

- MODINIT MODBUS initialization
- MODMAST MODBUS Master

3.1 Features of the communication 07 MK 92 R1161

The communication module 07 MK 92 R1161 is a freely programmable interface module with 4 serial interfaces.

Using this communication module, devices of other manufacturers can be coupled to the Advant Controller 31 system via the serial interface.

The communication protocols and the transfer rates can be freely defined by the user.

Programming is done using the programming and test software 907 MK 92 on a PC.

The communication module is connected to the Advant Controller 31 basic units 07 KR 91 R151 and 07 KT 9x via the networking interface.

The most essential features of the communication module are:

- 4 serial interfaces
where
 - 2 serial interfaces can be configured as desired according to EIA RS-232 or EIA RS-422 or EIA RS-485 (COM3, COM4) and
 - 2 serial interfaces can be configured according to EIA RS-232 (COM5, COM6).
- Freely programmable by means of an extensive function library.
- Communication to AC31 basic unit via connection elements.
- LEDs for diagnosis can be planned.
- Programming and testing on a PC using COM3.
- Data can be saved in a Flash EPROM.

The processing of the serial interfaces and the networking interface are planned in an application program.

The programming is performed using the high-level language "C".

The data exchange between the serial interface module and the Advant Controller 31 basic unit is realized in the basic unit using function blocks.

3.2 Notes concerning the programming software 907 MK 92

The programming software 907 MK 92 can be executed on normal IBM compatible personal computers having the following technical features:

- 640 kB RAM
- Hard disk drive
- Floppy disk drive, 3 ½", 1.44 Mbyte
- EIA RS-232 (V24) serial interface for the control system
- Parallel or second serial interface for printer
- MS-DOS operating system V5.0 or higher

Order notes:

907 MK 92

Programming and test software for the communication module 07 MK 92 R1161

Order number: GJP5 2073 00 R0102

→ no longer deliverable.

3.3 List of function blocks for the communication with the basic unit 07 MK 92

The function blocks are part of the programming software 907 AC 1131.

Library: ABB-BIB4.LIB or COM_S90_V41.LIB

The communication between the basic units 07 KT 9x and 07 MK 92 R1161 is performed using the following function blocks:

- SISEND Sending data to the networking interface
- SIREC Receiving data from the networking interface

4 Figures

Figure 1-1 RCOM network with multidrop modems	1-3
Figure 1-2 Event polling	1-5
Figure 1-3 Addressing in the RCOM system	1-7
Figure 1-4 Application of the connection elements for initialization	1-11
Figure 1-5 Application of the connection elements for a data transmission	1-12
Figure 1-6 Data set in which various data types are packed	1-14
Figure 1-7 Storing event data sets in the double word	1-15
Figure 1-8 Full-duplex data communication	1-19
Figure 1-9 Half-duplex data communication.....	1-20
Figure 1-10 Example of Setup file	1-26
Figure 1-11 Example telephone directory	1-28
Figure 1-12 Connecting Logem LGH 9600H1 to 07 KP 90 303	1-30
Figure 1-13 Setup 07 KP 90 R303	1-31
Figure 1-14 Settings LGH 9600H1	1-32
Figure 1-15 Modem cable	1-44
Figure 1-16 First variant of interface cable between modem and 07 KP 90	1-45
Figure 1-17 Second variant of interface cable between modem and 07 KP 90.....	1-45
Figure 1-18 First terminal cable	1-46

5 Tables

Table 1-1 Table of error codes	1-38
Table 1-2 Identification of services	1-39
Table 1-3 Significance of the message	1-40

O

07 KP 90 1-1
07 KP 93 2-1
07 MK 92 3-1

B

Broadcast 1-7

C

CALL_DELAY 1-27
Cold start 1-16
COMMAND 1-27
Commands
 DEBUG 1-36
 DIAL 1-36
 EVENT 1-35
 HANGUP 1-36
 HELP 1-35
 MOD 1-36
 PHONE 1-35
 RCOM 1-35
 RCSW 1-35
 SAVE 1-36
 SETUP 1-35
 SHOW PHONE 1-35
 SHOW SETUP 1-35
 TIME 1-35
Commissioning 1-8, 1-44
Console interface 1-33, 1-46

D

Data sets 1-4, 1-15, 1-25
Delay times 1-21
DIAL_PREFIX_1..6 1-27
DIAL_SUFFIX 1-27
Dial-up modem 1-24

E

Error codes 1-37
Event 1-25
Event-Polling 1-5

F

Full duplex 1-19

H

Half duplex 1-20
HANGUP_TIME 1-27

Hardware-Handshake 1-18

M

MAX_NO_OF_CALLS 1-27
MAX_RING_TIME 1-27
Modem parameters 1-26
MODEM_ANSWER 1-27
MODEM_HANGUP 1-27
MODEM_INIT 1-27
MODEM_RING 1-27

N

NO_CARRIER 1-27
Normalization 1-16

O

Operator commands *Siehe* Commands

P

Parameterization 1-46
Presetting 1-44

R

RCOM 1-3, 1-9
RCOM master 1-24
RCOM slave 1-25
RCOM system time 1-17
RCOM+ 1-9

S

Set clock 1-17
Setup 1-26

W

Warm start 1-16



ABB STOTZ-KONTAKT GmbH

Eppelheimer Straße 82 Postfach 101680
69123 Heidelberg 69006 Heidelberg
Germany Germany

Telephone +49 6221 701-0
Telefax +49 6221 701-240
E-Mail desst.help@de.abb.com
Internet <http://www.abb.de/stotz-kontakt>