



## Ethernet for the Saia PCD<sup>®</sup> series

**0 Table of contents**

0.1	Document History .....	0-4
0.2	Brands and trademarks .....	0-4

**1 Setup**

1.1	Important information .....	1-1
1.2	Configuring a Saia PCD® Ethernet port with Saia PG5® .....	1-1
1.2.1	Creating a new project containing all the CPUs on the Ethernet .....	1-1
1.2.2	Configuring the hardware settings .....	1-2
1.2.3	Downloading the configuration .....	1-3
1.3	Online connection via Ethernet .....	1-4
1.3.1	Selecting Online Settings .....	1-4
1.3.2	Establishing an online connection via Ethernet .....	1-4
1.4	Creating a user program in Fupla .....	1-6

**2 Hardware**

2.1	Saia PCD® systems with onboard Ethernet interface .....	2-1
2.1.1	PCD3.M3xx0 and PCD3.M5xx0 .....	2-1
2.1.2	PCD2.M5540 .....	2-2
2.1.3	PCD1.M2120 .....	2-2
2.1.4	PCD3.T665/T666 Ethernet RIO .....	2-2
2.2	Ethernet TCP/IP module PCD7.F65x .....	2-3
2.2.1	Block diagram .....	2-3
2.2.2	PCD7.F650/F655 Layout .....	2-4
2.2.3	PCD7.F651/F652 Layout .....	2-4
2.2.4	PGND connection .....	2-4
2.2.5	LED functions .....	2-5
2.2.6	RJ45 pin assignment .....	2-5
2.2.7	Cabling .....	2-6
2.2.8	Cable .....	2-7
2.3	Systems configured with PCD7.F65x .....	2-8
2.3.1	PCD1.M135F65x .....	2-8
2.3.2	PCD2.M150F65x .....	2-9
2.3.3	PCD2.M480F65x-2 .....	2-10
2.3.4	PCD4.M170Fx9 .....	2-11
2.3.5	PCD7.F65x on xx7 .....	2-11
2.4	Non-configured Ethernet-capable systems .....	2-12
2.4.1	PCD2.M170 with PCD7.F65x .....	2-12
2.4.2	PCD7.F65x on xx7 .....	2-12

**3 Attributes and functions**

3.1	Possible connections and network topologies .....	3-1
3.2	Ether-S-Bus .....	3-3
3.2.1	Network topology and addressing .....	3-4
3.2.2	Programming and troubleshooting via Ethernet .....	3-6
3.2.3	Multi-master communication .....	3-7
3.2.4	S-Bus gateway and S-Bus subnets .....	3-7
3.2.5	Rules for S-Bus gateway communication .....	3-8
3.2.6	Rules for OPC servers .....	3-9
3.2.7	Broadcast telegrams .....	3-11

3.3	Open Data Mode via TCP/IP or UDP/IP .....	3-13
3.4	Layout and structure of the network.....	3-15
3.4.1	Hubs (star network).....	3-15
3.4.2	Switches.....	3-15
3.4.3	Router .....	3-16
3.4.4	Network components .....	3-17
3.4.5	Increasing performance .....	3-18
<b>4</b>	<b>Configuration and programming</b>	
4.1	Configuration and addressing .....	4-1
4.1.1	Configuring the S-Bus IP port (server) .....	4-1
4.1.2	Addressing the IP server station .....	4-2
4.1.3	Slots and channel numbers .....	4-3
4.2	Programming the S-Bus via Ethernet .....	4-4
4.2.1	Description of Saia PCD® commands .....	4-4
4.2.2	Using Saia PG5® FBoxes to program the Ethernet S-Bus.....	4-11
4.2.3	S-Bus IP multimaster .....	4-14
4.2.4	Broadcast telegrams via S-Bus IP .....	4-14
4.3	Programming the Open Data Mode via Ethernet.....	4-15
4.3.1	Description of Open Data Mode.....	4-15
4.3.2	Configuration.....	4-15
4.3.3	Important characteristics of UDP and TCP in Open Data Mode.....	4-16
4.3.4	Programming with IL .....	4-17
4.3.5	InitODM.....	4-21
4.3.6	Diagnostics .....	4-25
4.3.7	Amount of ODM channels.....	4-29
4.3.8	OpenUDP.....	4-30
4.3.9	OpenClientTCP.....	4-31
4.3.10	OpenServerTCP .....	4-32
4.3.11	Close.....	4-33
4.3.12	ConnectTCP .....	4-34
4.3.13	DisconnectTCP.....	4-35
4.3.14	GetConnectionTCP .....	4-36
4.3.15	AcceptConnectionTCP .....	4-37
4.3.16	SendData .....	4-38
4.3.17	SendDataRev .....	4-39
4.3.18	ReceiveData .....	4-40
4.3.19	ReceiveDataRev .....	4-41
4.3.20	Byte swapping .....	4-42
4.3.21	IP address decoding .....	4-43
4.3.22	A typical TCP connection process .....	4-44
4.4	Additional CSFs .....	4-45
4.4.1	CSF NA Reset .....	4-45
4.4.2	CSF SetLocalIPNode .....	4-45
4.4.3	CSF IPPhyConfig .....	4-46
4.4.4	CSF SendEtherSBUS .....	4-48
4.4.5	CSF RecvEtherSBUS .....	4-50
4.5	Ethernet TCP/IP error messages.....	4-51

**5 Diagnostics and troubleshooting**

5.1	Summary.....	5-1
5.2	The TCP/IP communication process.....	5-1
5.3	The top-down rule.....	5-1
5.3.1	PING.....	5-2
5.3.2	ARP.....	5-3
5.3.3	Other useful commands for the host computer.....	5-3
5.4	Wireshark Ethernet protocol analyzer.....	5-4
5.5	PCD7.F655 IP stack debugging via RS-232 on the Saia PCD® .....	5-9

**6 Programming examples**

6.1	Reference to Ethernet links.....	6-1
6.2	Performance testing.....	6-1
6.3	Example program: TCP/IP Open Data Mode.....	6-2
6.3.1	Server .....	6-3
6.3.2	Client.....	6-7

**A Appendix**

A.1	Icons .....	A-1
A.2	Contact .....	A-2



## 0.1 Document History

Version	Date	Released	Remarks
EN01	-	-	First edition
EN02	2002-06-02	2002-06-02	Table addressing RxBroadcast_error (Bit14) and text corrections
EN03	2003-08-03	2003-08-03	Number of channels allowed Communication data
EN05	2010-01-15	2010-02-17	Translated from DE05
EN06	2011-07-21	2011-07-21	Chapter 2: Added full-duplex and MDIX
EN07	2012-04-04	2012-04-04	Chapter 4.3.9: “OpenUDP” replaced with “OpenClientTCP”
EN08	2013-11-05	2013-11-05	New logo and new company name
EN09	2014-02-18 2014-04-03	2014-04-03 2014-04-03	Ch4 - flow chart replaced Ch4 - length of transmitted user data UDP
ENG10	2016-02-26	2016-02-26	Ch4.2.1 Error in „Example of use“
ENG11	2017-09-01	2017-09-01	all “IPD” replaced with “IP”
ENG12	2018-06-05	2018-06-05	Ch2.2.5 - Table comments
ENG13	2019-08-08	2019-08-08	Typo corrected “CTP” -> “TCP”

## 0.2 Brands and trademarks

Saia PCD® and Saia PG5®  
are registered trademarks of Saia-Burgess Controls AG.

Technical modifications are based on the current state-of-the-art technology.

Saia-Burgess Controls AG, 2002 © All rights reserved.

Published in Switzerland

# 1 Setup

## 1.1 Important information

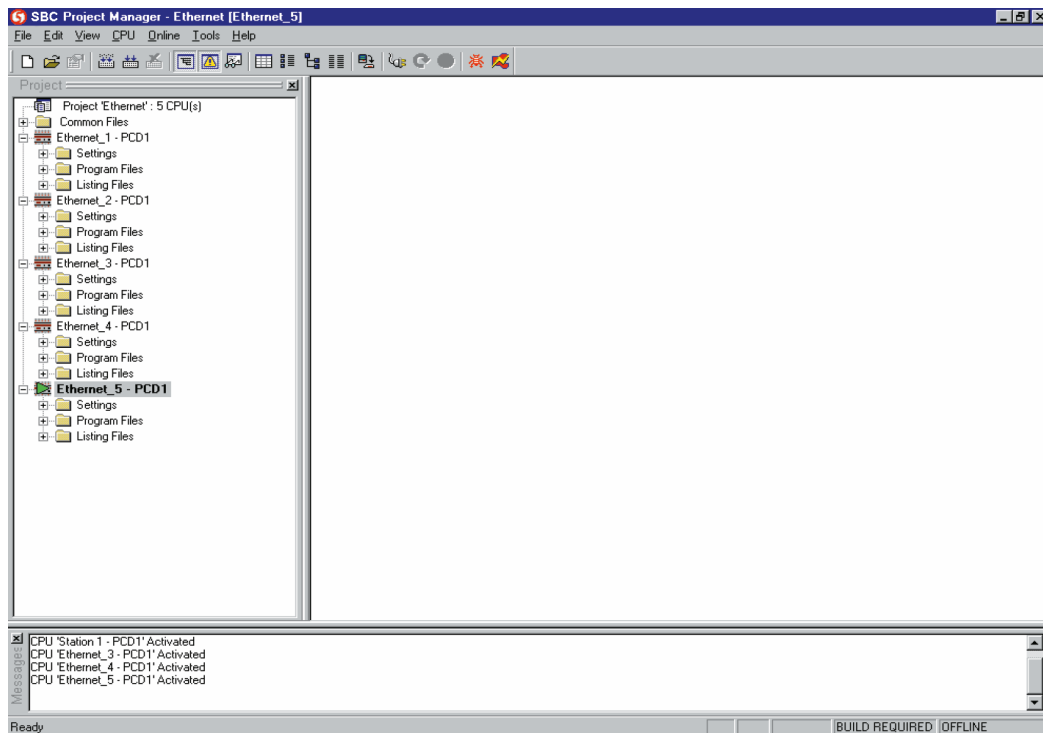
1

- Apart from on the PCD2.M480, only one Ethernet module can be used per Saia PCD® system
- Only one S-Bus PGU port is permitted per system. If the Ethernet module is configured as an S-Bus PGU, no other port can be used as an S-Bus PGU port
- The Ethernet module can also be configured for the S-Bus reduced protocol (see the guidelines which follow). In this case, any other port may be used as an S-Bus PGU.
- An Ethernet station requires an allocation table with the IP address and the IP node for all stations on the network. This allocation table is generated automatically by the hardware configurator and loaded as a DBX into the Saia PCD®. The hardware configuration of all PCD systems on the Ethernet must therefore be defined within the same project in the Project Manager.
- If several people are programming different Saia PCD® stations in the same project, it is recommended that the entire network configuration first be defined on a single programmer PC. This configuration should then be copied onto the other programmer PCs or imported from the programmer PCs.
- Gateway communication is only possible in the Ethernet to S-Bus subnet direction, not in the opposite direction. Only one S-Bus subnet is permitted per gateway.

## 1.2 Configuring a Saia PCD® Ethernet port with Saia PG5®

You can find more information in the Saia PG5® help file.

### 1.2.1 Creating a new project containing all the CPUs on the Ethernet



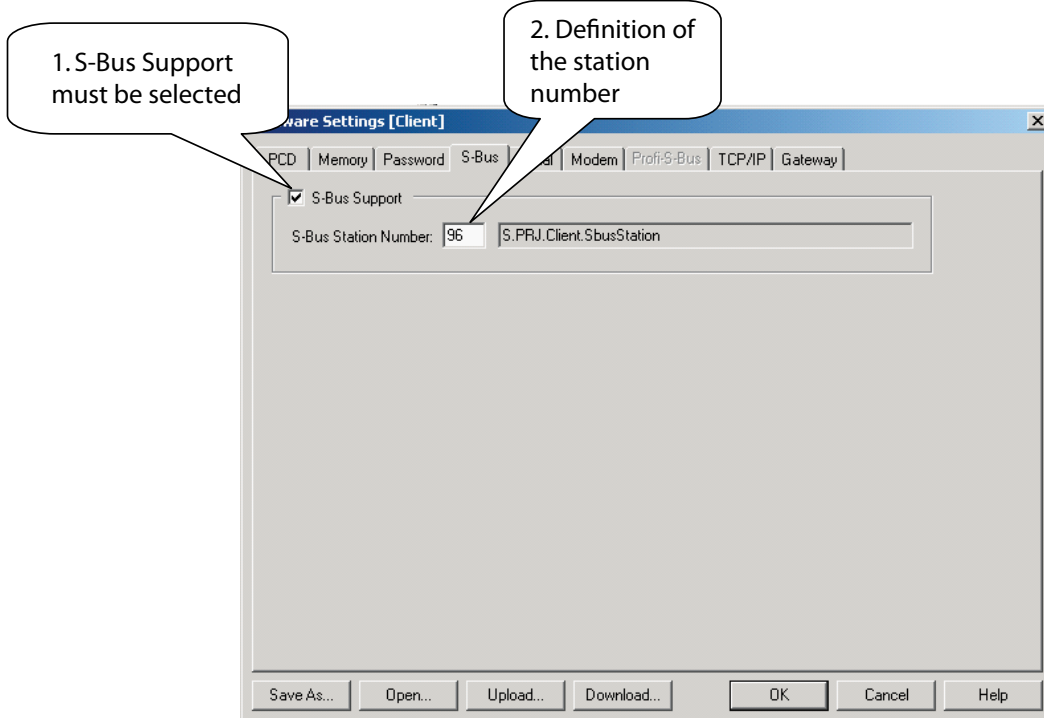
Hardware settings must be specified for all Saia PCD® stations.

### 1.2.2 Configuring the hardware settings

The following windows are required for configuring the Ethernet:

1

- **SBC S-Bus settings**

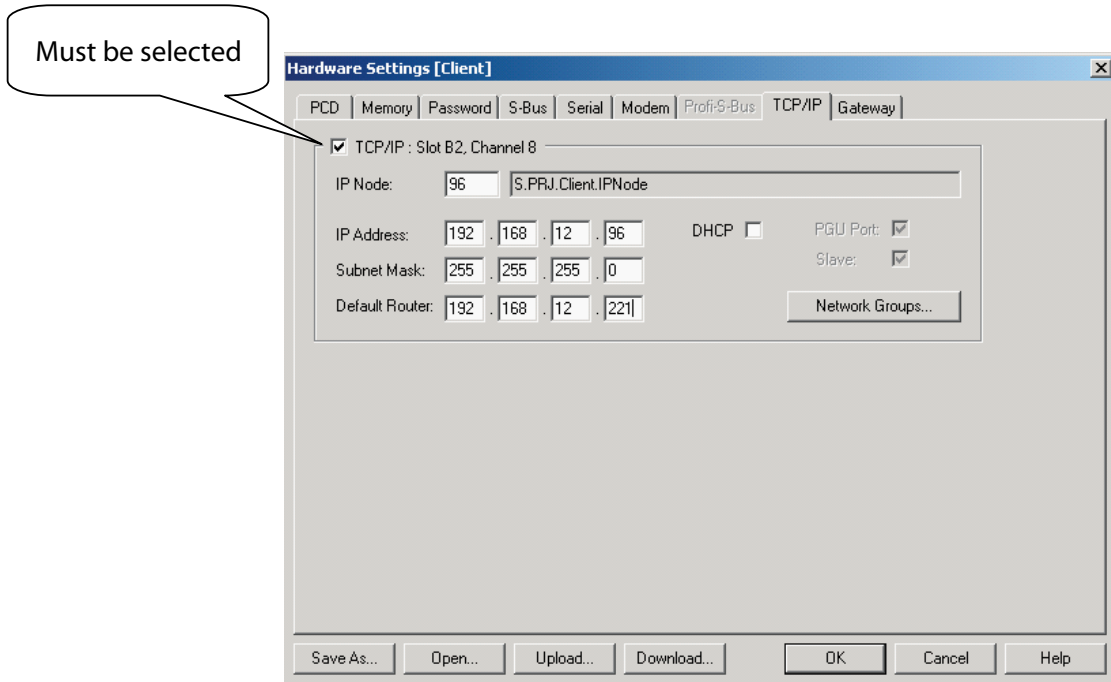


- **Gateway settings**

The definition is necessary only for a gateway station.

The settings are the same as those for standard S-Bus communication.

- **Settings for the Ethernet TCP/IP module**



Enter the values for IP node, IP address, subnet mask and default router.

Where there is only one subnet, it is not strictly necessary to enter the default router details.

Channel and slot settings depend on where the Ethernet module has been installed.

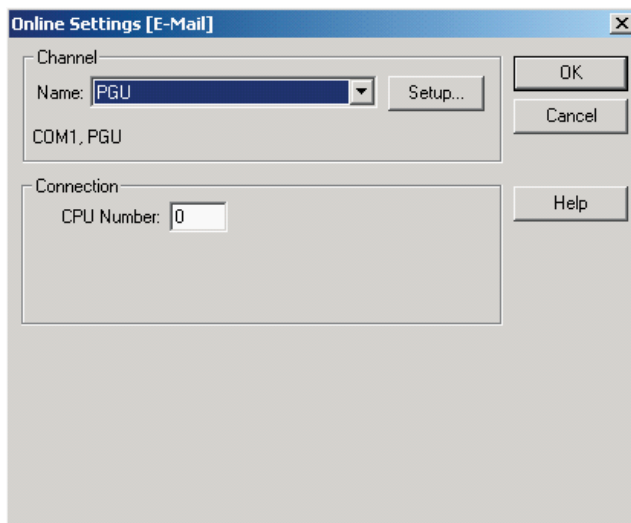
For simple applications, it is not necessary to modify the Network Groups setting.

You can also find detailed information in the Saia PG5® help file.

1

### 1.2.3 Downloading the configuration

Initially, the configuration must be downloaded via the PGU port, using the PGU channel.

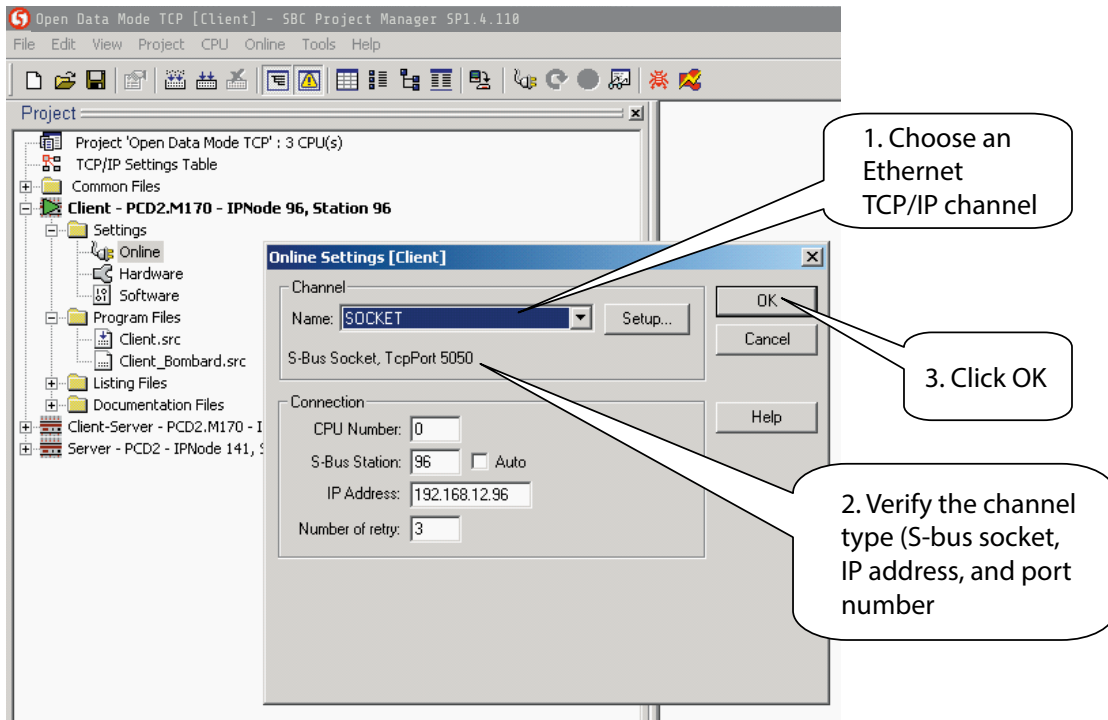


After selecting the PGU channel, the configuration settings can be downloaded from the Hardware Settings menu.

### 1.3 Online connection via Ethernet

#### 1.3.1 Selecting Online Settings

1

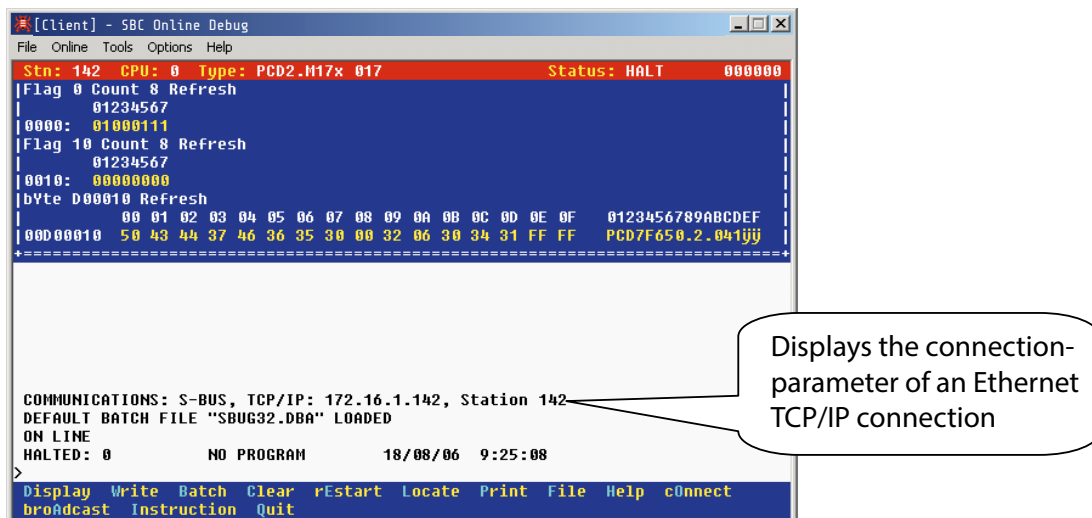


The values for CPU number, S-Bus station and IP address are set automatically in accordance with the hardware settings for the selected station; however, they can be adjusted if required.

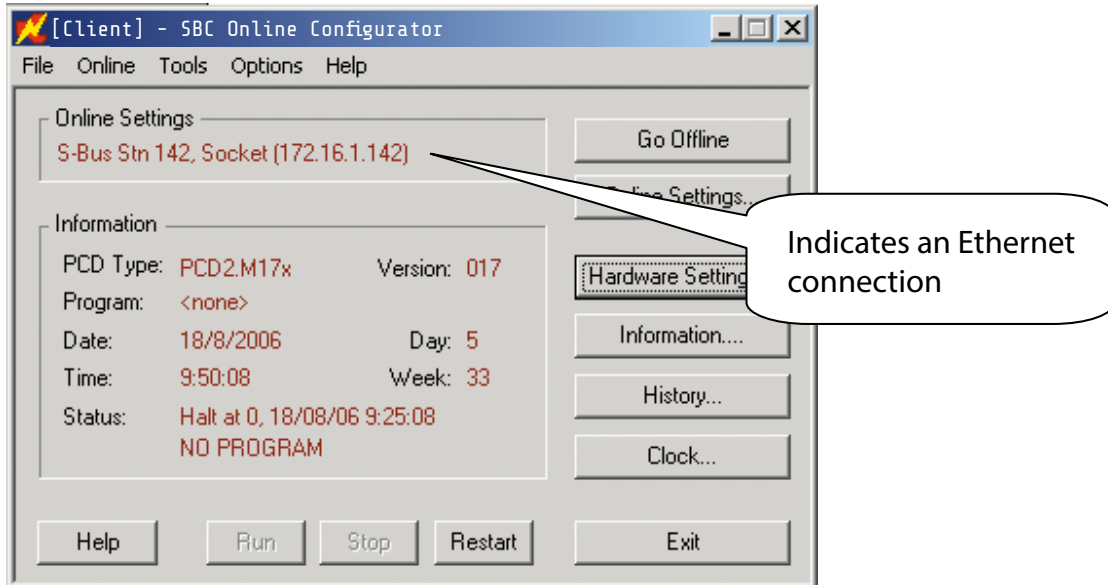
#### 1.3.2 Establishing an online connection via Ethernet

The connection to the Saia PCD®s can be established using one of the following online tools.

Online connection using the debugger:

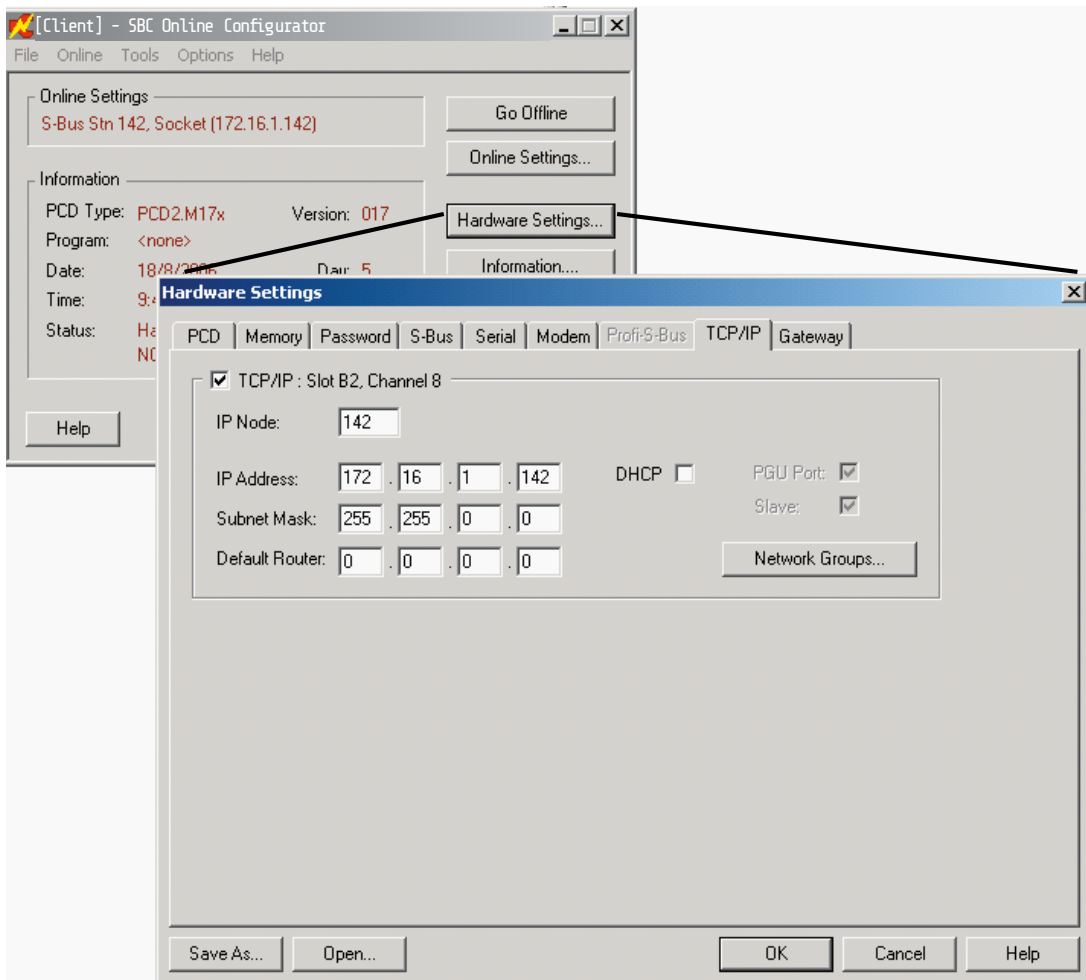


Online connection using the online configurator:



1

The settings for Ethernet TCP/IP in the Saia PCD® can also be checked or modified using the following menu commands:



If the online connection does not work properly despite the Saia PCD®'s being configured correctly, see Chapter 5: Diagnostics and Troubleshooting.

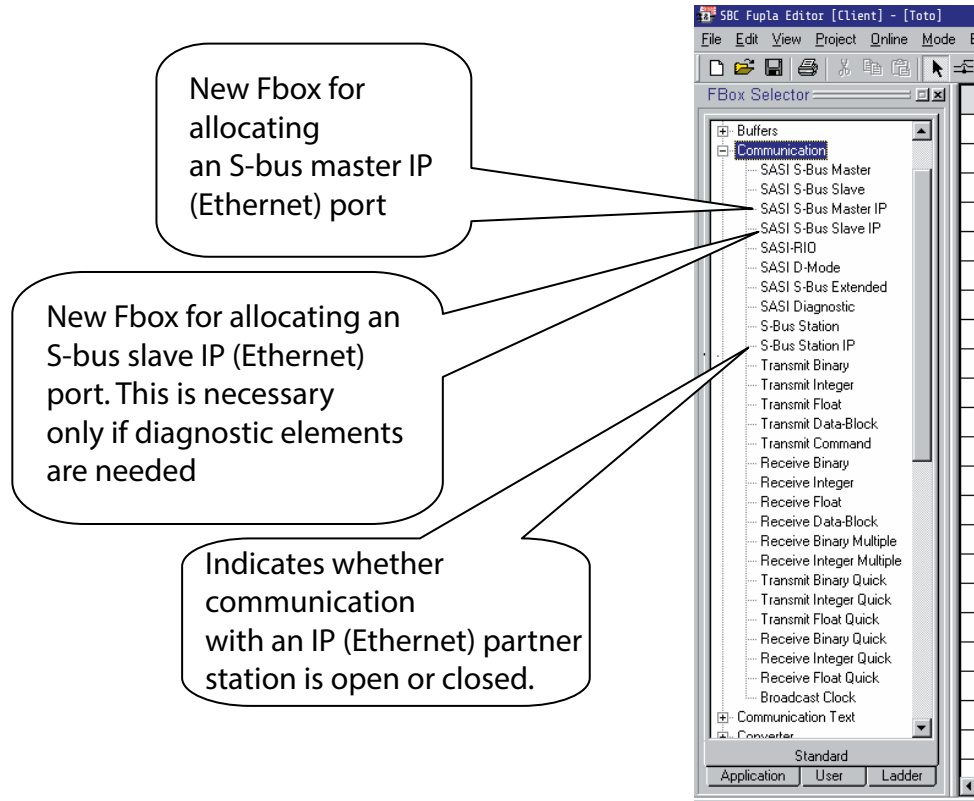
### 1.4 Creating a user program in Fupla

In order to program a communication event via Ethernet, an up-to-date communication library is required.

Ethernet TCP/IP is supported beginning with version \$2.2.003.

Check that the new library has been installed correctly.

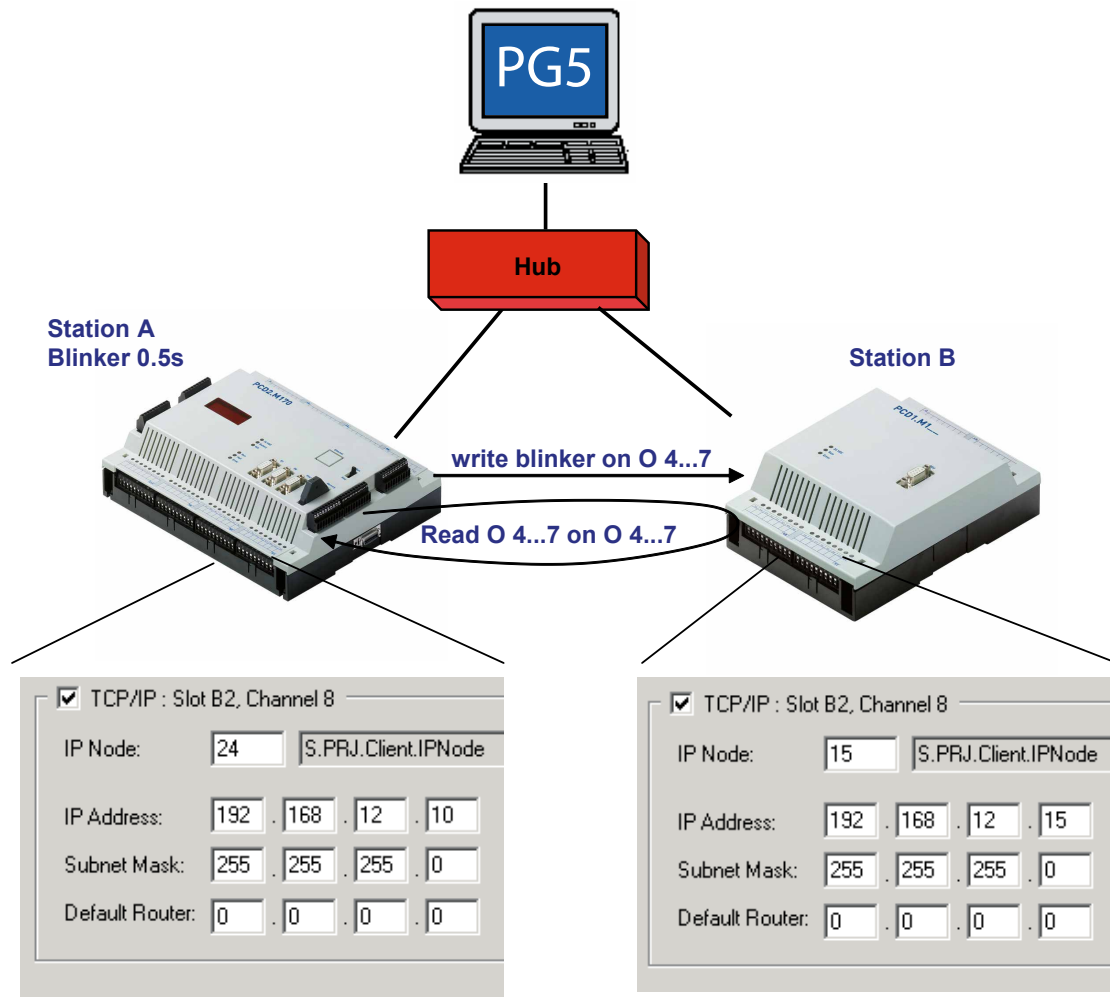
An additional parameter must be defined for all Send/Receive Saia PG5® FBoxes (for example, Send/Receive Binary): the IP Node Number.



Ethernet TCP/IP communication events are programmed in the same way as a standard S-Bus communication event.

**Programming example in Fupla:**

1



**Task:**

A flashing signal is generated at Station A. The flashing signal is copied via Ethernet TCP/IP to Station B on outputs 4-7. These outputs are read back and copied to outputs 4-7 of Station A.



User program in Fupla:

The screenshot shows the SBC Fupla Editor interface. On the left is the 'FBox Selector' with a tree view of communication-related functions. The main workspace displays a ladder logic diagram with several blocks: 'S-Bus Master IP', 'SASI-Diag', 'SEND', and 'RCV'. A 'Blink' block is also present. A dialog box titled 'Adjus: Senden Binär' is open, showing configuration parameters for a binary send operation.

Group/Symbol	Type	Address/Value	Comment
out4	Output	4	
out5	Output	5	
out6	Output	6	
out7	Output	7	

Dialog box 'Adjus: Senden Binär' configuration:

- Initialization: No
- Node: 15
- Destination station: 3
- Destination element: Output
- Destination address: 4

Page Navigator: COB COB Pag

Status bar: Type = Senden Binär: Name = : Factor = 3 [0..19] | Block: COB COB\_3AC1AA2F | Page: 1/1 [58x54] | FR | OFFLINE

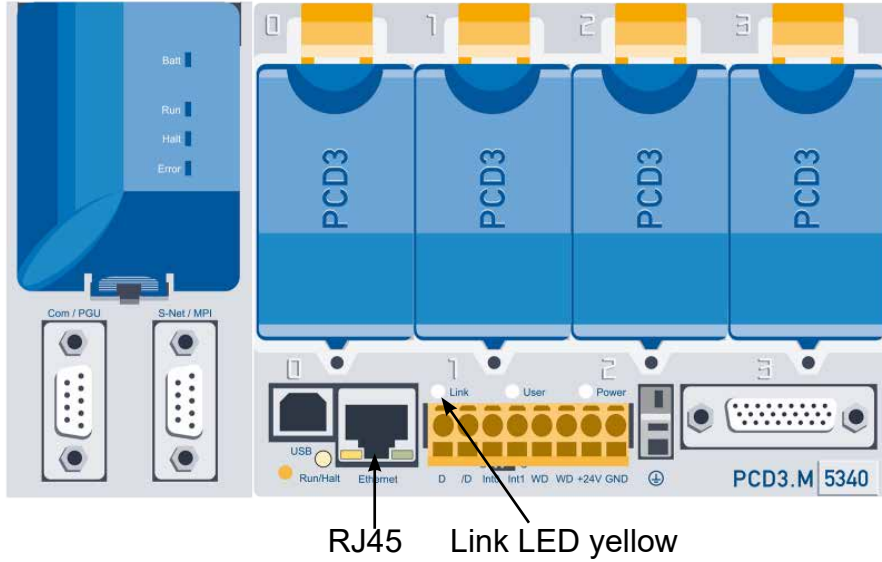
1

## 2 Hardware

### 2.1 Saia PCD® systems with onboard Ethernet interface

The PCD3.Mxx family has members with onboard Ethernet interface. Ethernet traffic uses channel 9 on these devices.

2



#### 2.1.1 PCD3.M3xx0 and PCD3.M5xx0

PCD3.M3xx0 and PCD3.M5xx0 with hardware version F and more recent support full-duplex mode and Auto-MDIX (auto-crossing the signals). Whether a PCD3.Mxxx0 supports full-duplex and Auto-MDIX can be seen by checking the RJ45 for LEDs. If the connector is equipped with LEDs, full-duplex and Auto-MDIX is supported.



#### Order information:

PCD3.M5540	Ethernet-capable PCD3	
PCD3.M3120	Ethernet-capable PCD3	(without battery, not extendable)
PCD3.M3330	Ethernet-capable PCD3	(without battery, extendable I/Os)
PCD3.M6340	Ethernet-capable PCD3	(with LAN interface)
PCD3.M6540	Ethernet-capable PCD3	(with Profibus DP master interface)
PCD3.M2330A4Tx	PCD3 Wide Area Controller (WAC)	
PCD3.M2130V6	PCD3 Compact	
PCD3.M5560	PCD3 Power CPU	
PCD3.M6360	PCD3 Power CPU	(with LAN interface)
PCD3.M5560	PCD3 Power CPU	(with Profibus DP master interface)

### 2.1.2 PCD2.M5540

The **PCD2.M5540** features an integrated Ethernet switch with two connectors. The Ethernet communication uses channel 9 on the PCD2.M5540. Full-duplex mode and Auto-MDIX (auto-crossing the signals) are supported by the PCD2.M5540 by all hardware versions.

### 2.1.3 PCD1.M2120

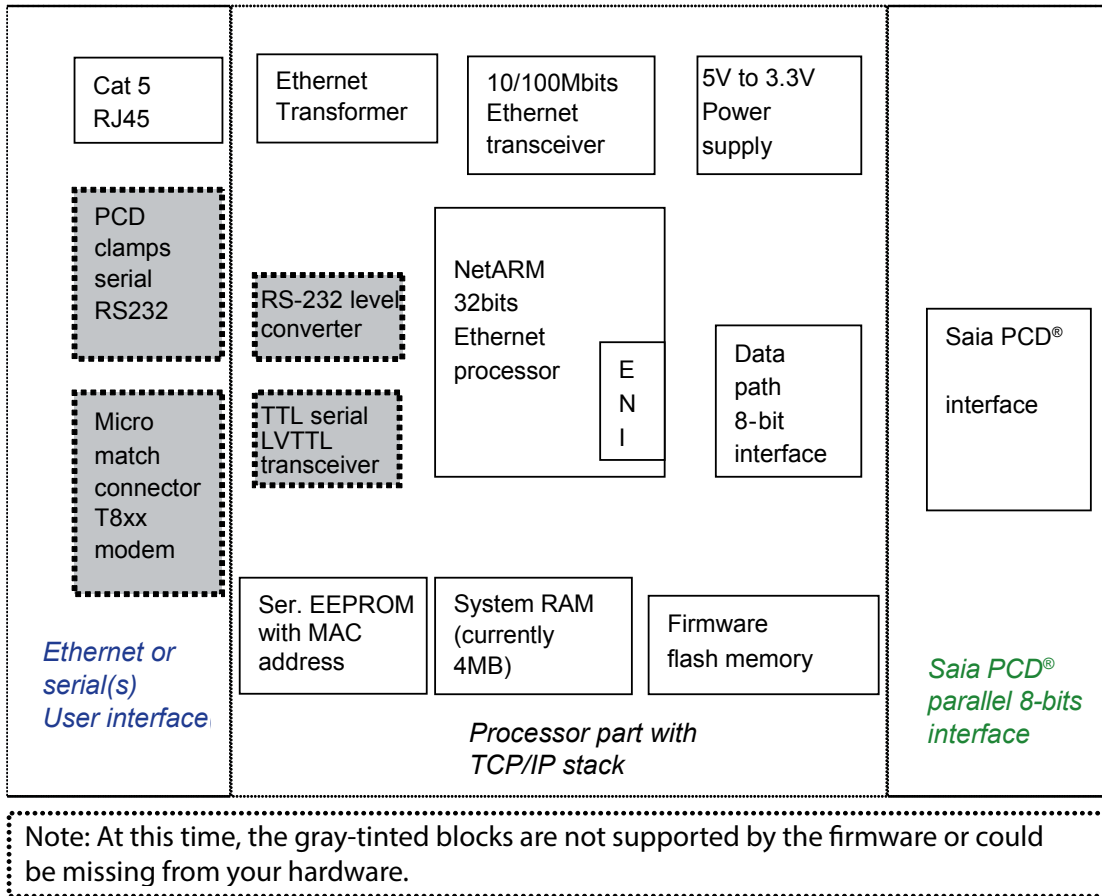
The **PCD1.M2120** features an integrated Ethernet switch with two connectors. The Ethernet communication uses channel 9 on the PCD1.M2120. Full-duplex mode and Auto-MDIX (auto-crossing the signals) are supported by the PCD1.M2120 by all hardware versions.

### 2.1.4 PCD3.T665/T666 Ethernet RIO

The **PCD3.T665/T666 Ethernet RIO** features an integrated Ethernet interface with two connectors. The Ethernet communication uses channel 9 on the PCD3.T665/T666. Full-duplex mode and Auto-MDIX (auto-crossing the signals) are supported by the PCD3.T665/T666 by all hardware versions.

**2.2 Ethernet TCP/IP module PCD7.F65x**

**2.2.1 Block diagram**

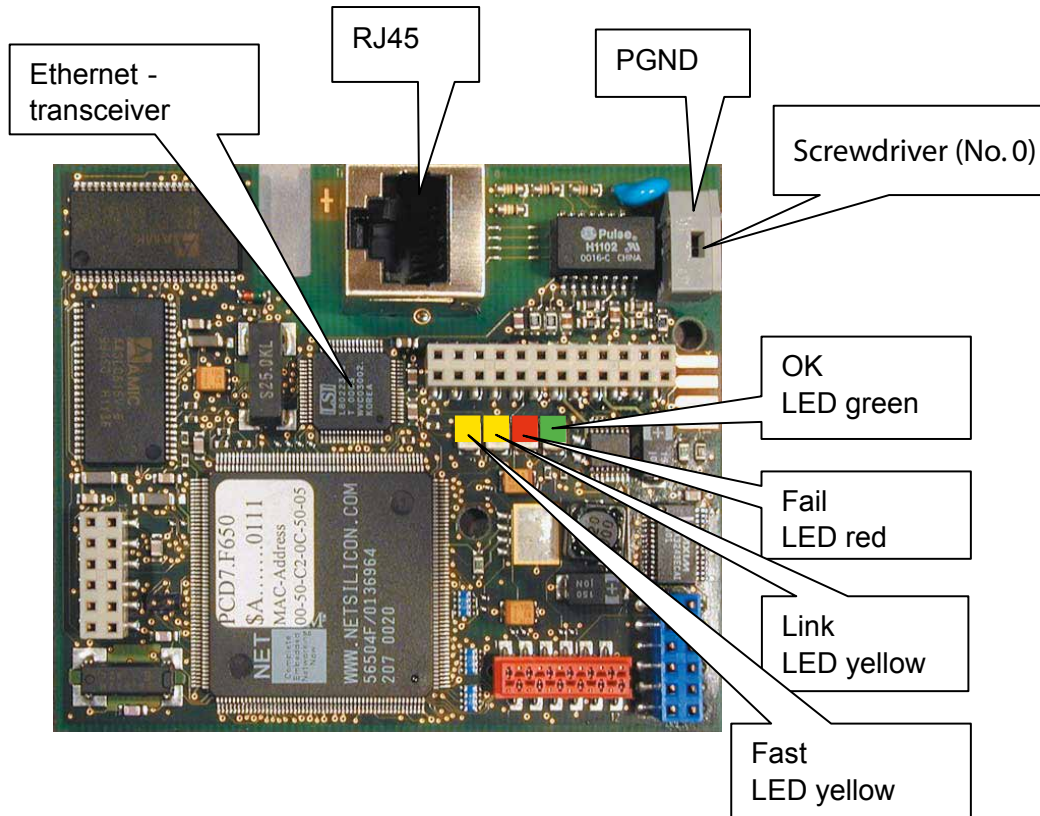


2



The serial interfaces are managed by the Ethernet processor and cannot be used as regular Saia PCD® interfaces. These two interfaces are intended for IP connections (PPP, SLIP, etc.) via modem or for troubleshooting purposes.

## 2.2.2 PCD7.F650/F655 Layout



## 2.2.3 PCD7.F651/F652 Layout

The layout corresponds to that of the PCD7.F650/F655, with the exception of the RJ45 connector, which comes in several variants.

## 2.2.4 PGND connection

To maintain the EMC values, the PGND (protective ground) must always be connected to the ground connection of the frame.

### To connect the protective ground:

- Open the spring-force terminal block with a No. 0 screwdriver (no. 0) inserted in the slot.
- Insert the ground wire into the PGND port
- Remove the screwdriver again
- Tug at the protective ground lead to check that it is properly connected

### 2.2.5 LED functions

- Fast (yellow) This LED lights up when a fast, 100 Mbit/s connection is detected, and remains unlit for 10 Mbit/s
- Link (yellow) This LED lights up when a connection is detected and flashes to indicate Ethernet traffic (off for activity, on for no activity)
- Fail (red) This LED flashes if a hardware or firmware error is detected
- OK (green) This LED indicates that there are no faults in operation (flashes with a frequency of 2 Hz)



During the processor boot sequence, the yellow LEDs have a different meaning than during normal operation. Immediately after a reset (during switch-on), the Fast LED shows a 10 Mbit/s connection, while the Link LED indicates a 100 Mbit/s connection.

Link LED yellow	Fast LED yellow	Comment
LINK 100	LINK 10	After reset
LINK + ACT	10/100	During operation

The Fail LED (red) indicates that an error has been detected. The rate of flashing provides information about the error which has been detected. The OK LED may also flash.

Depending on the error detected, the green and red LEDs may flash. If the green LED is flashing on its own, there is no error.



Risk of injury! The Ethernet transceiver gets quite hot. When not connected, its temperature is about 30 °C above the ambient temperature. This is normal.

### 2.2.6 RJ45 pin assignment

Pin	Name	Twisted pair	Wire colour	Ethernet	Fast Ethernet	Note
1	TPO+	Pair 2	white/orange	TPO+	TPO+	
2	TPO-	Pair 2	orange	TPO-	TPO-	
3	TPI+	Pair 3	white/green	TPI+	TPI+	
4	Term. 1a	Pair 1	blue	-	Pair 1	terminated with 75 Ω*
5	Term. 1b	Pair 1	white/blue	-	Pair 1	together with Pin 4*
6	TPI-	Pair 3	green	TPI-	TPI-	
7	Term. 4a	Pair 4	white/brown	-	Pair 4	terminated with 75 Ω*
8	Term. 4b	Pair 4	brown	-	Pair 4	together with Pin 8*

\* Bob Smith termination

For Ethernet (10 Mbit/s), Pair 1 and Pair 4 can be absent.

Fast Ethernet cable (CAT5 cable) is compatible with 10 Mbit/s Ethernet.

## 2.2.7 Cabling

### Cable type:

This module is designed for use both with shielded and non-shielded 100-Ohm cable (UTP or STP). If there is electromagnetic interference, shielded cable is highly recommended for higher throughput. If a 100 Mbit/s connection is used, CAT 5 cable must be used as a minimum. For a 10 Mbit/s connection, CAT 3 cable must be used as a minimum.



This module uses an auto-negotiation procedure to determine the transfer speed and operating mode. In order to establish a 100 Mbit/s, full-duplex connection, both ends must support auto-negotiation. If this is not the case, the module will establish a 10 Mbit/s connection in half-duplex mode. However, the auto-negotiation procedure cannot detect the type of cable being used. This means that, while a 100 Mbit/s connection can in fact be established over a CAT 3 cable, such a connection will probably not work reliably (this may occur with a crossover cable, for example).

A special Call System Function (CSF) makes it possible to switch the PCD7.F65x into full-duplex or half-duplex mode and to use either 10 Mbit/s or 100 Mbit/s for communications. You can find more details on this in the chapter concerning the special CSFs.

### Cable length:

Maximum 100 m

Bending radius:

Certain restrictions apply to bending radius.

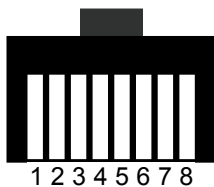
In accordance with EIA SP-2840A, the minimum bending radius is 10 times the outside diameter of the cable. In accordance with ISO DIS 11801 it is 8 times the outside diameter.

### Wiring:

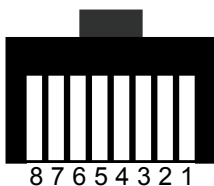
The network can be wired either in a peer-to-peer fashion using a crossover cable, or using a normal patch cable through a hub or switch.

### Pin allocation:

**Connector** (view of the connector end, cable running away from the observer)



**Socket** (looking at the wall socket)



## 2.2.8 Cable

**Cable configuration for crossover cable (peer-to-peer):**

Pin	Colour		Pin
1	or/wh	←→	3
2	orange	←→	6
3	gn/wh	←→	1
4	blue	←→	4
5	wh/bl	←→	5
6	gn	←→	2
7	wh/br	←→	7
8	brown	←→	8

2

**Cable configuration for patch cable (via a hub):**

Pin	Colour		Pin
1	or/wh	←→	1
2	orange	←→	2
3	gn/wh	←→	3
4	blue	←→	4
5	wh/bl	←→	5
6	gn	←→	6
7	wh/br	←→	7
8	brown	←→	8

Colours defined in accordance with EIA/TIA T568B.

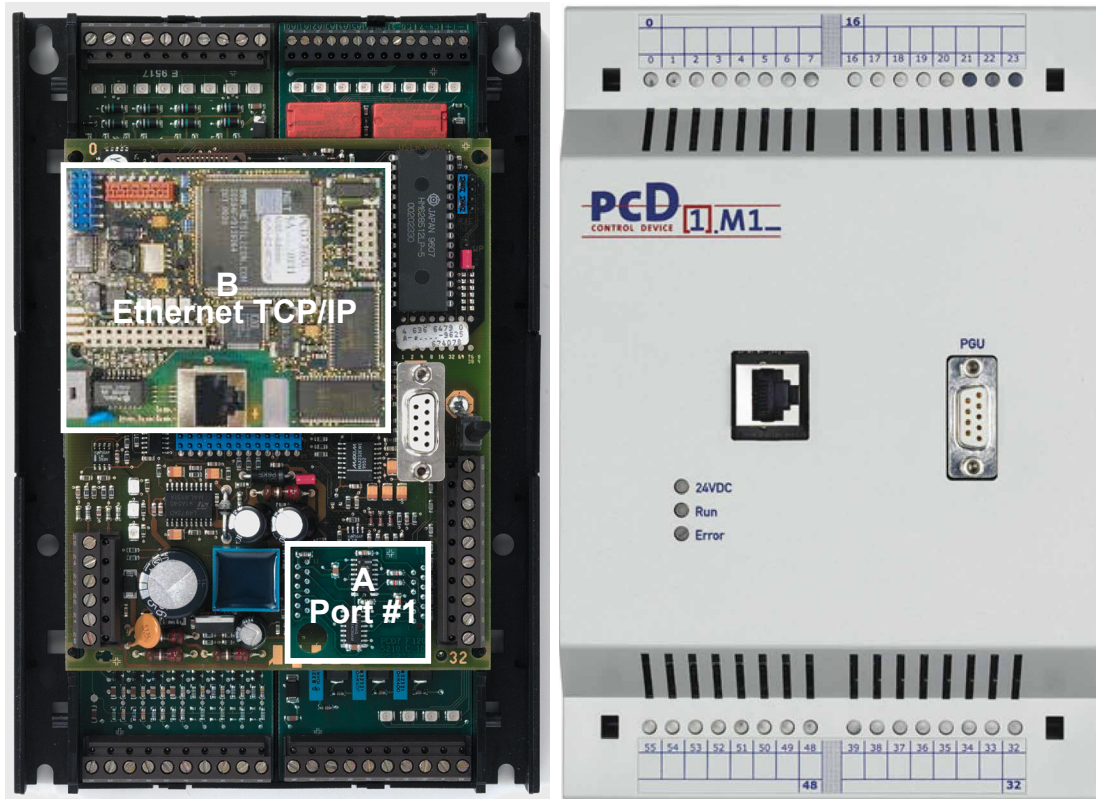


**2.3 Systems configured with PCD7.F65x**

**2.3.1 PCD1.M135F65x**

PCD7.F65x in Slot B / Channel 9 PCD1.M135F65x

2



The PGND lead from the PCD7.F65x must be connected to terminal 23 of the PCD1.

**Order information:**

- As a configured system:
 

PCD1.M135F655	PCD1 configured system with Ethernet module
---------------	---
  
- As an add-on:
 

PCD7.F655	Ethernet module for PCD1/PCD2
4'104'7409'0	Cover for PCD1.M135 with cut-out for RJ45 connector

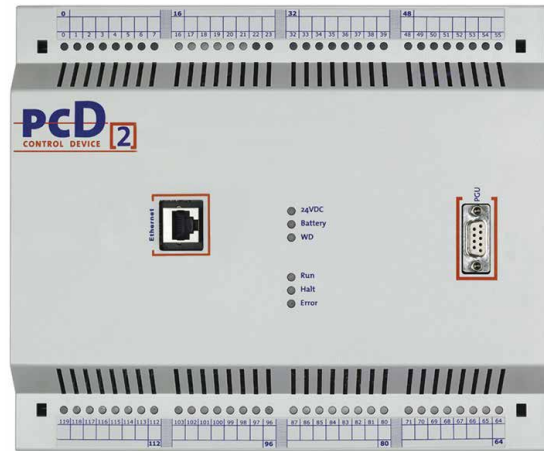
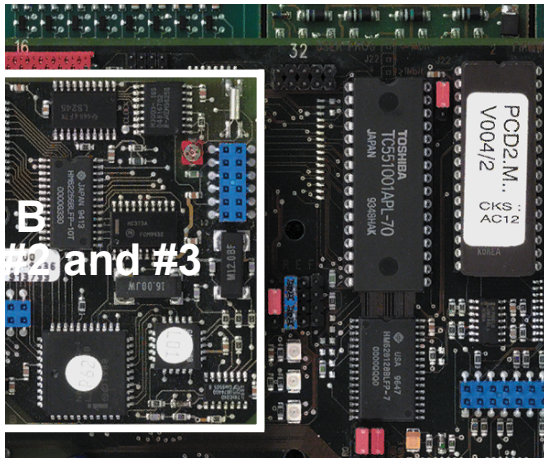


There may be restrictions with older versions of the PCD1 hardware.

**2.3.2 PCD2.M150F65x**

PCD7.F65x in Slot B / Channel 9

PCD2.M150F65x



2



The PGND lead from the PCD7.F65x must be connected to terminal 27 of the PCD2. Saia PCD® terminals 30-39 are reserved for RS-232 (RS-232 is in the pipeline).

**Order information:**

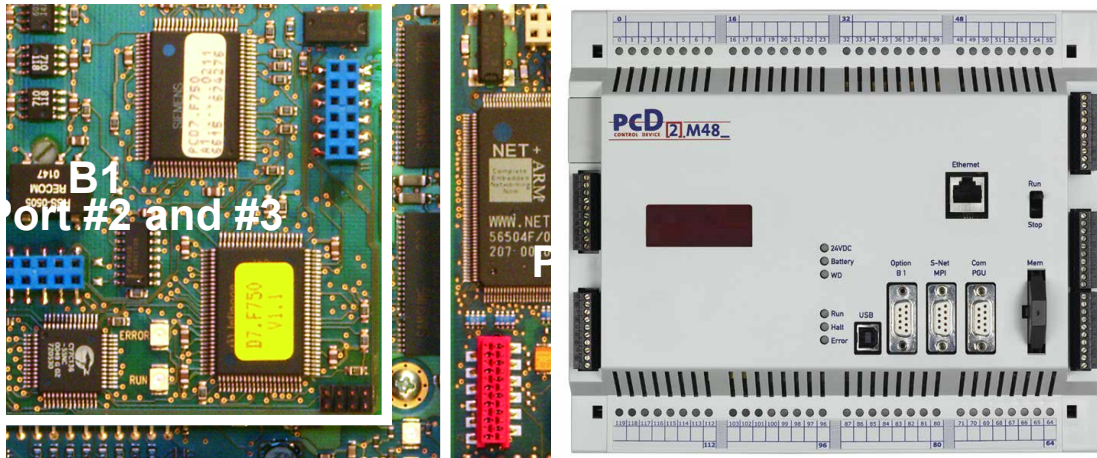
- As a configured system:
 

PCD2.M150F655	PCD2 configured system with Ethernet module
---------------	---
  
- As an add-on:
 

PCD7.F655	Ethernet module for PCD1/PCD2
4'104'7410'0	Cover for PCD2.M150 with cut-out for RJ45 connector

**2.3.3 PCD2.M480F65x-2**

PCD7.F65x in Slot B2 / Channel 8 and also in Slot B1 / Channel 9



The PGND lead from the PCD7.F65x must be connected to the PGND terminal next to B1 DB9. Saia PCD® terminals 40-49 are reserved for RS-232 (RS-232 is in the pipeline).

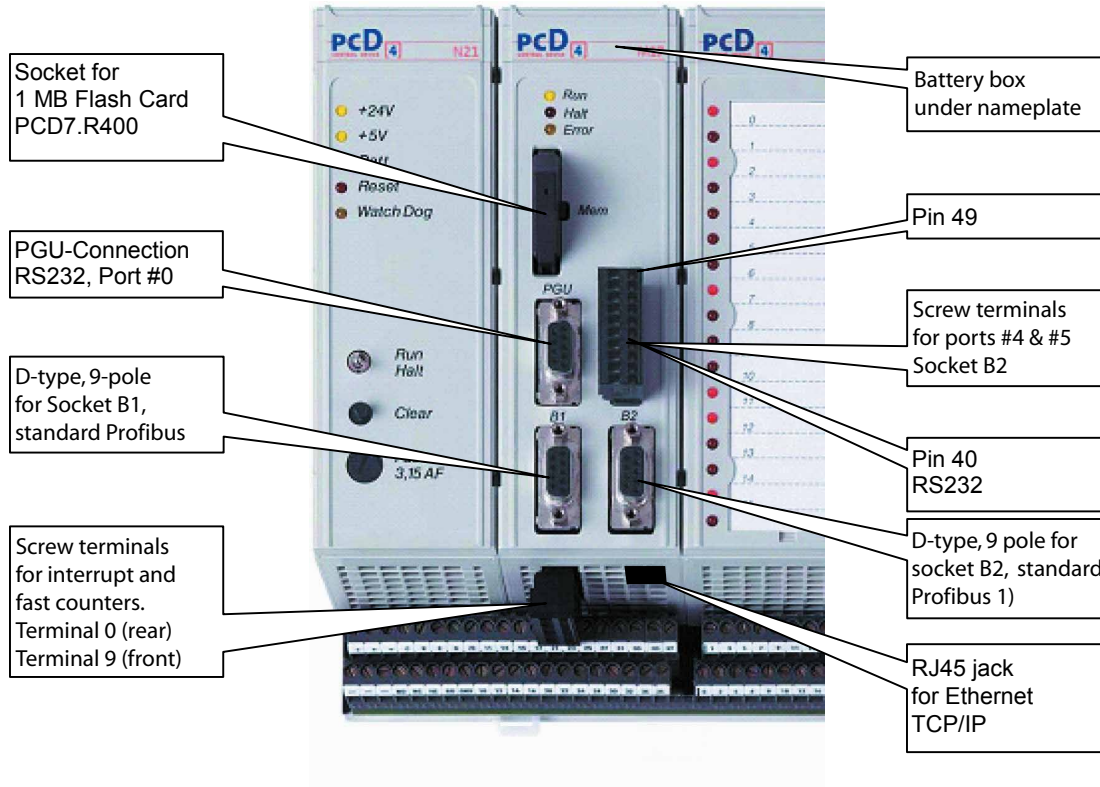
**Order information:**

- As a configured system:  
PCD2.M480F655-2 PCD2 configured system with two Ethernet modules
  
- As an add-on:  
PCD7.F655 Ethernet module for PCD1/PCD2  
4'104'7503'0 Cover for PCD2.M480F655-2 with cut-out for two RJ45 connector

**2.3.4 PCD4.M170Fx9**

The Ethernet connection is made via an RJ 45 connector attached to the bottom of the PCD4.M17x. Only available as a PCD4.M170Fx9 configured system (where x is the code which describes the interface plugged into B1).

PCD7.F65x in Slot B1 / Channel 9



The PGND lead of PCD7.F65x must be connected to PGND. Saia PCD® terminals 40-49 are reserved for RS-232 (RS-232 is in the pipeline).

**Order information:**

- As a configured system:  
 PCD4.M170Fx9      PCD4 configured system with Ethernet module

**2.3.5 PCD7.F65x on xx7**

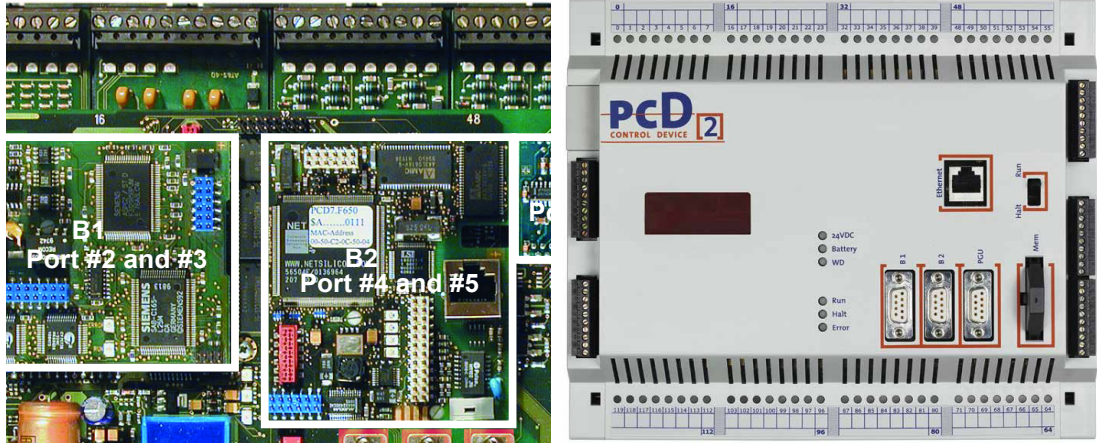
For the Ethernet-capable xx7 with PCD7.F65x, both configured and non-configured systems are available. For information on this, please refer to the xx7 Ethernet manual 26-791.



**2.4 Non-configured Ethernet-capable systems**

**2.4.1 PCD2.M170 with PCD7.F65x**

PCD7.F65x in Slot B2 / Channel 8



The PGND lead of PCD7.F65x must be connected to the PGND terminal next to B1 DB9. Saia PCD® terminals 40-49 are reserved for RS-232 (RS-232 is in the pipeline).

**Order information:**

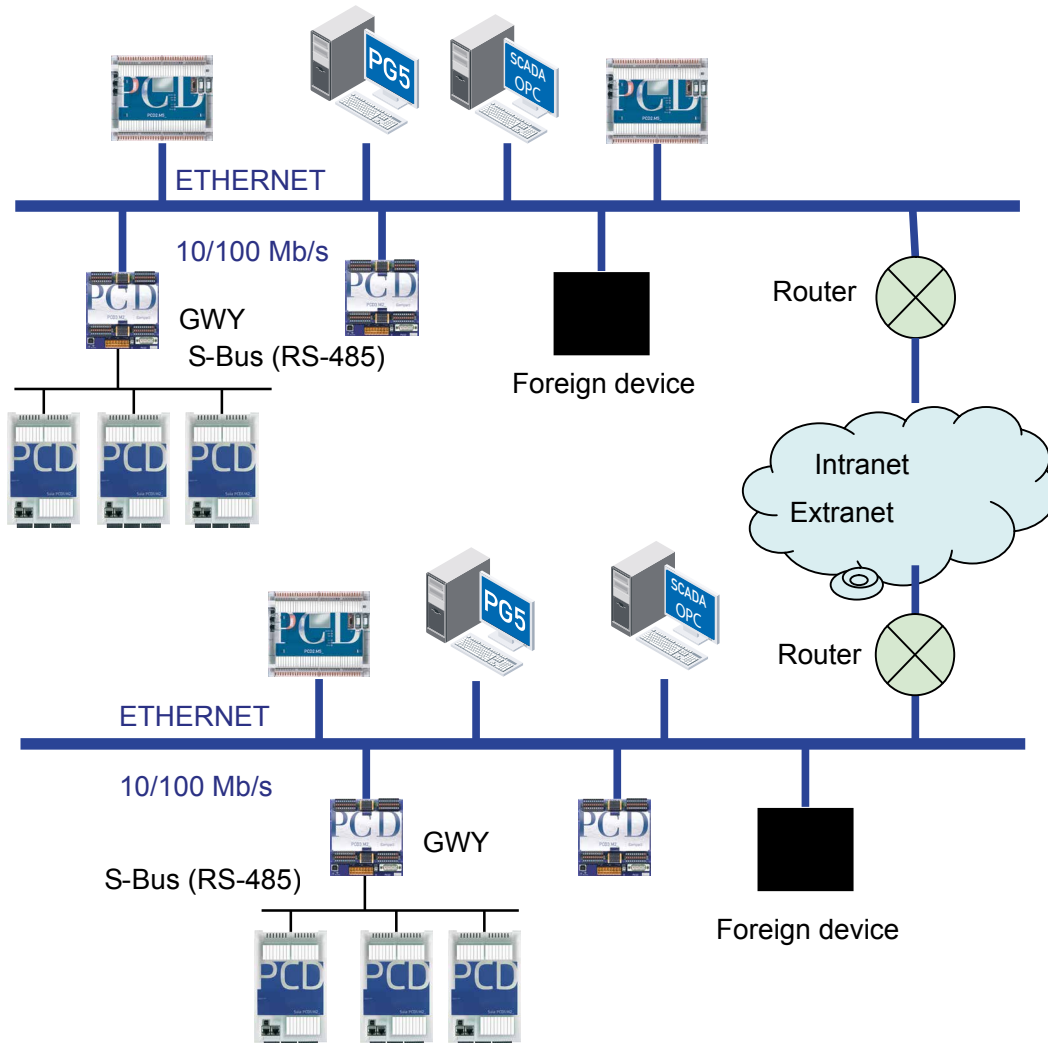
PCD7.F655                      Ethernet module for PCD1/PCD2

**2.4.2 PCD7.F65x on xx7**

For the Ethernet-capable xx7 with PCD7.F65x, both configured and non-configured systems are available. For information on this, please refer to the xx7 Ethernet manual 26-791.

### 3 Attributes and functions

#### 3.1 Possible connections and network topologies



3

An Ethernet normally allows all types of connections between stations connected to the network. The Saia PG5<sup>®</sup> programming tool and a SCADA system can access a PCD and a station from a third-party manufacturer directly via the Ethernet. The Saia PG5<sup>®</sup> programming tool and the SCADA system, which is based on SCOM.dll, are clients.

Ethernet TCP/IP modules from Saia Burgess Controls do not require a proprietary network and can be used on standard networks with standard components such as hubs, switches, routers, etc. The modules support all current network topologies.

An S-Bus network can be set up and placed under a gateway station connected to the Ethernet. The PCD on this S-Bus subnet is accessed indirectly via a PCD which is configured as the Ethernet gateway station. This allows it to forward messages received from the Ethernet to the subordinate S-Bus network. This makes it easy to integrate multiple S-Bus networks into the Ethernet.

Two protocols are used on an Ethernet TCP/IP network:

- S-Bus protocol (UDP/IP, port 5050)
- Open Data Mode protocol (UDP/IP or TCP/IP and user-defined ports), which uses a socket interface to implement a user protocol.

### Overview of functions and how to achieve them

Functions	Achieved using		
	S-Bus Ethernet UDP/IP Port 5050	Open Data Mode Ethernet UDP/IP	Open Data Mode Ethernet TCP/IP
Programming of the PCD and troubleshooting with Saia PG5®	✓		
PCD multi-master S-Bus communication	✓		
Connecting PCD stations to a SCADA system	✓	✓	✓
Communication between PCD stations and a third-party system			
Implementation of user protocols		✓	✓
S-Bus gateway functionality (from the Ethernet to the standard S-Bus)	✓		

S-Bus UDP, Open Data Mode UDP, and Open Data Mode TCP can be operated simultaneously using the same PCD.

### 3.2 Ether-S-Bus

The Ether-S-Bus protocol is used for communication between:

- two PCDs
- one PCD and the Saia PG5® programming tool
- one PCD and other stations (SCADA systems, OPC servers or another PLC which support the Ether-S-Bus protocol).

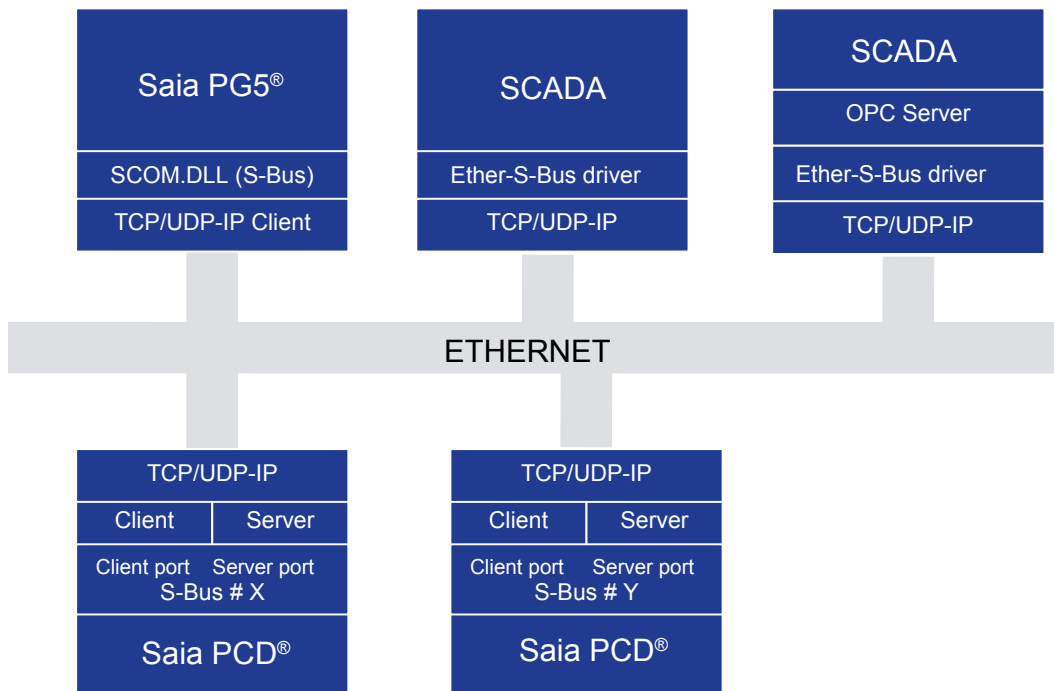
Data is exchanged using the well-known STXM/SRXM commands in IL or the convenient FUPLA FBoxes. The syntax is very similar to existing S-Bus telegrams.

S-Bus via IP is implemented using UDP sockets via the fixed **input** port 5050. When managing connections through firewalls, you should ensure that this port is **enabled** in the firewall configuration.

Multi-master communication between PCDs is supported by Ether-S-Bus. For this reason, every PCD has a server and a client port, and is able to simultaneously work as a client or remain passive as a server station.

#### Diagram of the application layers

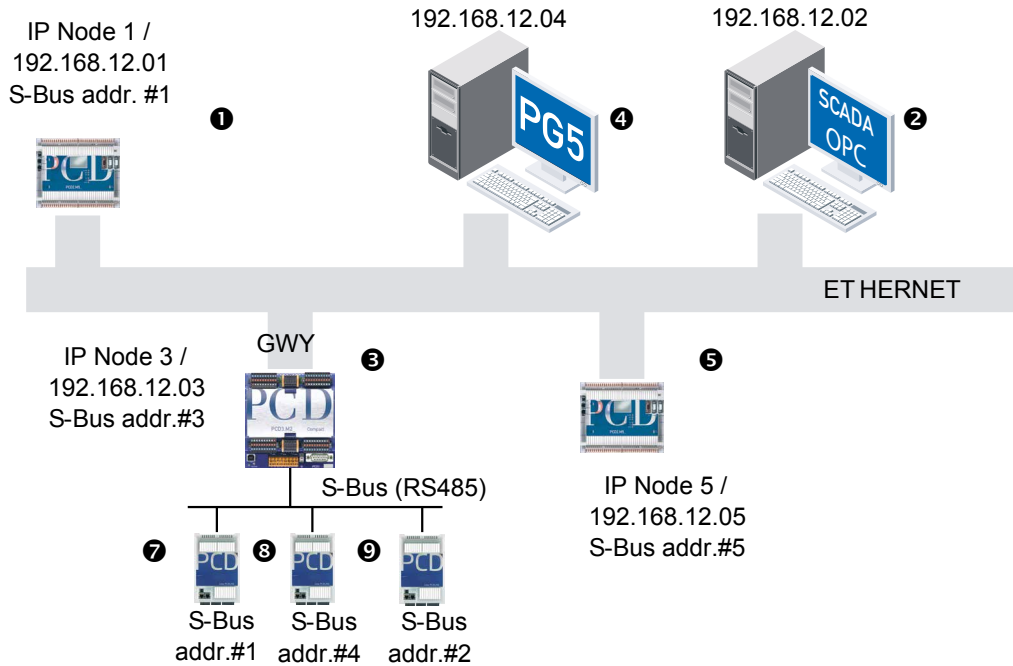
- Saia PG5® based on SCOM.dll
- SCADA system supporting Ether-S-Bus
- SCADA system connected to an OPC
- PCD with Ether-S-Bus and server ports



A SCADA system can either support Ether-S-Bus, or it can connect to the SBC OPC server, which in turn is supporting Ether-S-Bus.



**3.2.1 Network topology and addressing**



Each IP module on the Ethernet is identified by a fixed MAC address, which is unique in the world. The station is then configured with an IP address. This is often location-dependent within the company, and is not registered on the internet (private IP address). An IP node must be defined according to SBC's specifications.

The conversion table (MAC address ↔ IP address) is managed internally by the TCP/IP and ARP stacks.

In order to make address handling as easy as possible for the user, an additional layer of abstraction has been introduced with the IP node. The IP module is configured in the Saia PG5® hardware configurator by the assignment of an IP address, a single IP node, and an S-Bus address. The same number is often used for a station's S-Bus address and IP node.

Later on, when the program is running, only the IP node number and the S-Bus address of the station are used for communication on the Ethernet. After a station has been configured, the user no longer has to deal with the IP address.

A table containing all the combinations of IP addresses, IP nodes, and S-Bus addresses for the whole project is generated by Saia PG5® and saved in the PCD.

Table of the stations from the perspective of the PCD, from IP node #1 upwards (IP address 192.168.12.01, S-Bus address #1).

Station	IP address	IP node	S-Bus address
5	192.168.12.05	5	5
3	192.168.12.03	3	3
7	192.168.12.03	3	1
8	192.168.12.03	3	4
9	192.168.12.03	3	2



You can find more details on the handling of the address table in the Saia PG5® help file: Hardware Settings - TCP/IP page.

Before the STXM/SRXM command is called, the destination address must be loaded into the address register, as is the case with any standard S-Bus protocol. As is illustrated in the following examples, addressing uses two address fields.

3

Station 5	LDL R 100 5 ;S-Bus address LDH R 100 5 ;IP node  STXM 9 10 F 500 O 32
Station 3	LDL R 100 3 ;S-Bus address LDH R 100 3 ;IP node  STXM 9 10 F 500 O 32
Addressing unit 3	LDL R 100 4 ;S-Bus address LDH R 100 3 ;IP node  STXM 9 10 F 500 O 32

### 3.2.2 Programming and troubleshooting via Ethernet

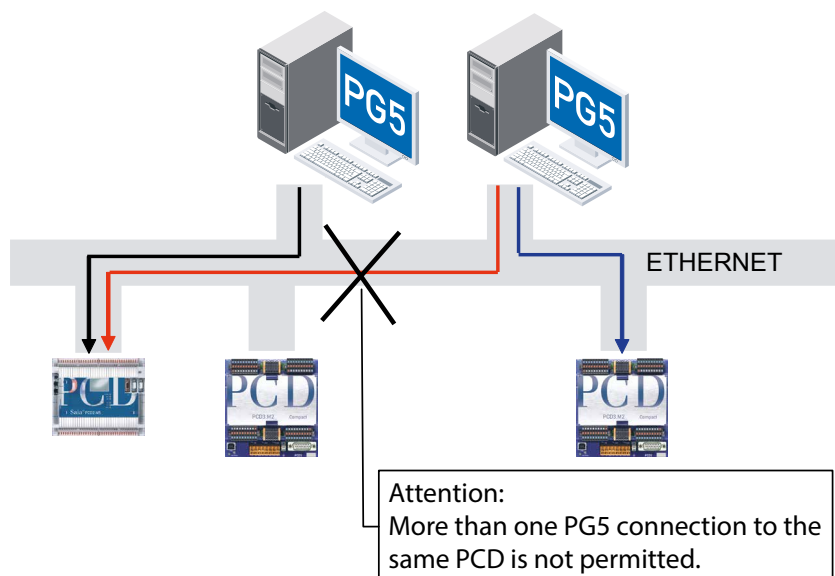
Configuring, programming, downloading, and debugging a PCD station can be performed via Ethernet using the Saia PG5<sup>®</sup> programming tool and the well-known S-Bus protocol. Program downloads are much faster via Ethernet than using the S-Bus over a serial interface. This is particularly useful for very large user programs.

Programming via Ethernet using the S-Bus protocol is supported beginning with Saia PG5<sup>®</sup> version 1.1. This uses:

- the instruction list (IL) or
- convenient FUPLA FBoxes.

Data is exchanged within the IL via the usual STXM/SRXM commands. The syntax is very similar to standard S-Bus telegrams. Access security is provided by standard S-Bus password protection.

It is possible to use more than one Saia PG5<sup>®</sup> programming tool on an Ethernet. Doing so allows the parallel development and implementation of large projects by several programmers or by teams in order to save time. However, only one S-Bus UDP full-protocol connection to the PCD is permitted at a time.

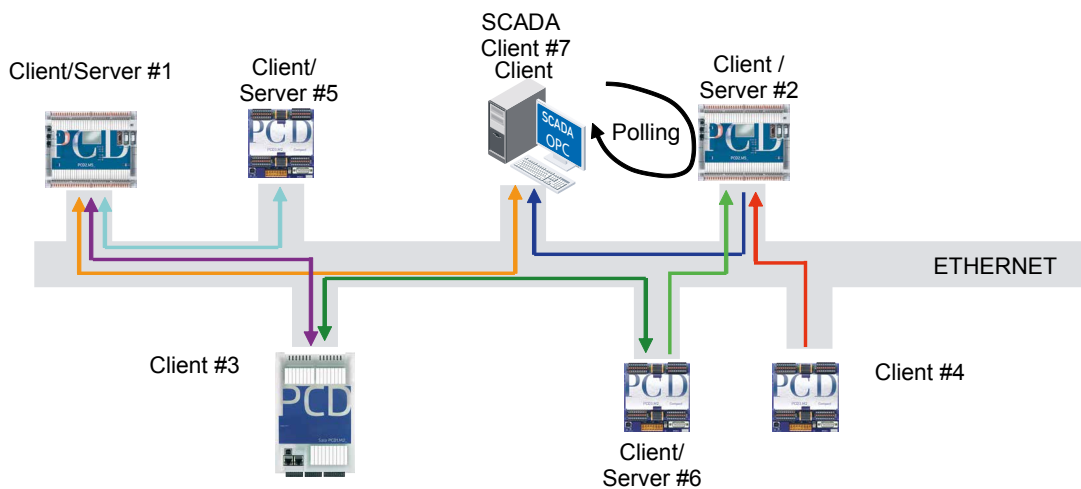


### 3.2.3 Multi-master communication

Compared to the standard S-Bus protocol, the new master/master functionality for Ethernet is a considerable improvement. On the standard S-Bus via RS-485, only one client was allowed per network. With S-Bus via Ethernet, all stations are able to work as clients or servers.

By designing the network for multi-master operation, it is possible to establish powerful, event-driven communication connections between PCD stations. This explains why each PCD has both a client port and a server port.

Therefore a PCD (Client/Server #2), for instance, is able to receive information from several other PCDs (Client/Server #6 and Client #4) and be queried periodically by a SCADA system or an OPC server.



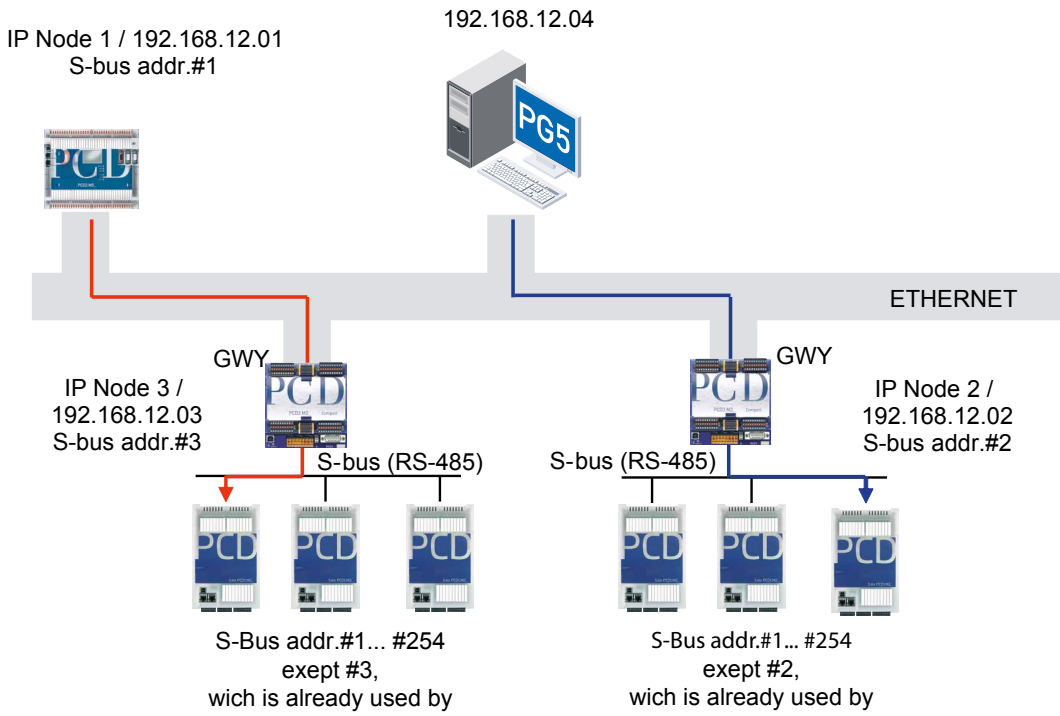
### 3.2.4 S-Bus gateway and S-Bus subnets

An S-Bus subnet can be set up under a gateway station (GWY) connected to the Ethernet.

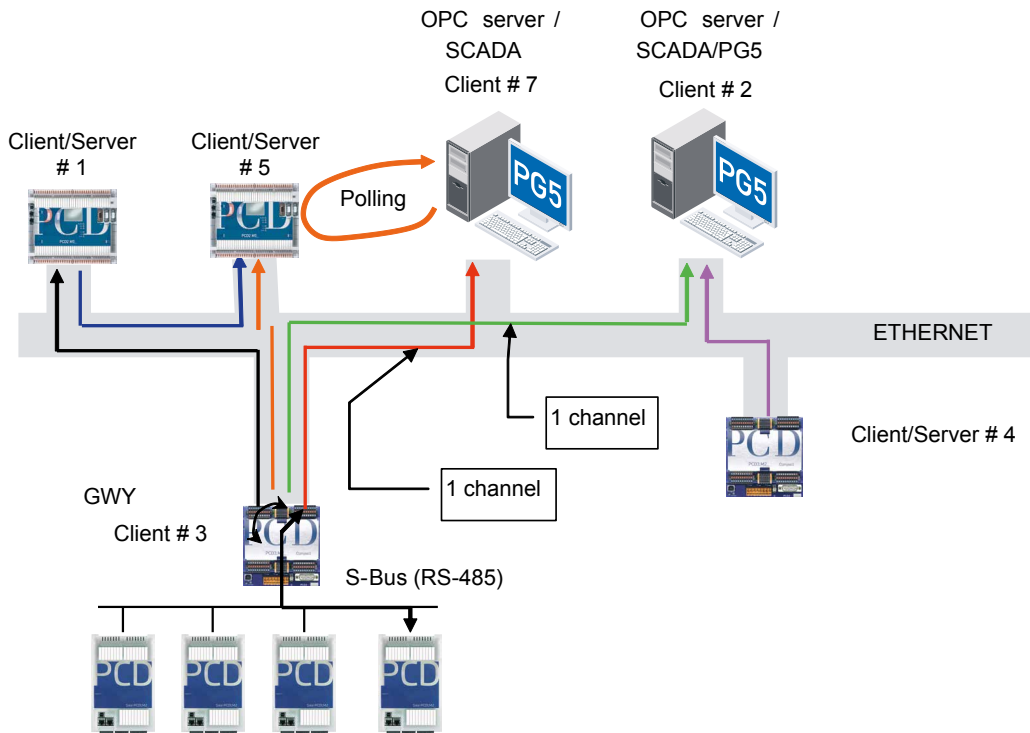
Access to the PCDs on this S-Bus subnet occurs indirectly via the PCD, which is configured as the Ethernet gateway station (GWY). This allows it to forward messages received from the Ethernet to the subordinate S-Bus network. This gateway station is the only client station allowed on the S-Bus subnet.

This makes it easy to integrate multiple S-Bus networks into the Ethernet. It is therefore possible to address 65,535 IP nodes or 65,535 x 254 PCDs (254 PCDs on an S-Bus subnet, including the PCD configured as the gateway station) on an Ethernet.

A PCD on an S-Bus subnet only requires an S-Bus address (not an IP address).



**3.2.5 Rules for S-Bus gateway communication**



A limited number of buffers is reserved in a gateway station for handling incoming telegrams.

PCD1 GWY:	1 buffers	(= 1 telegram; max. 1 client)
PCD2 GWY:	4 buffers	(= 4 telegrams; max. 4 clients)

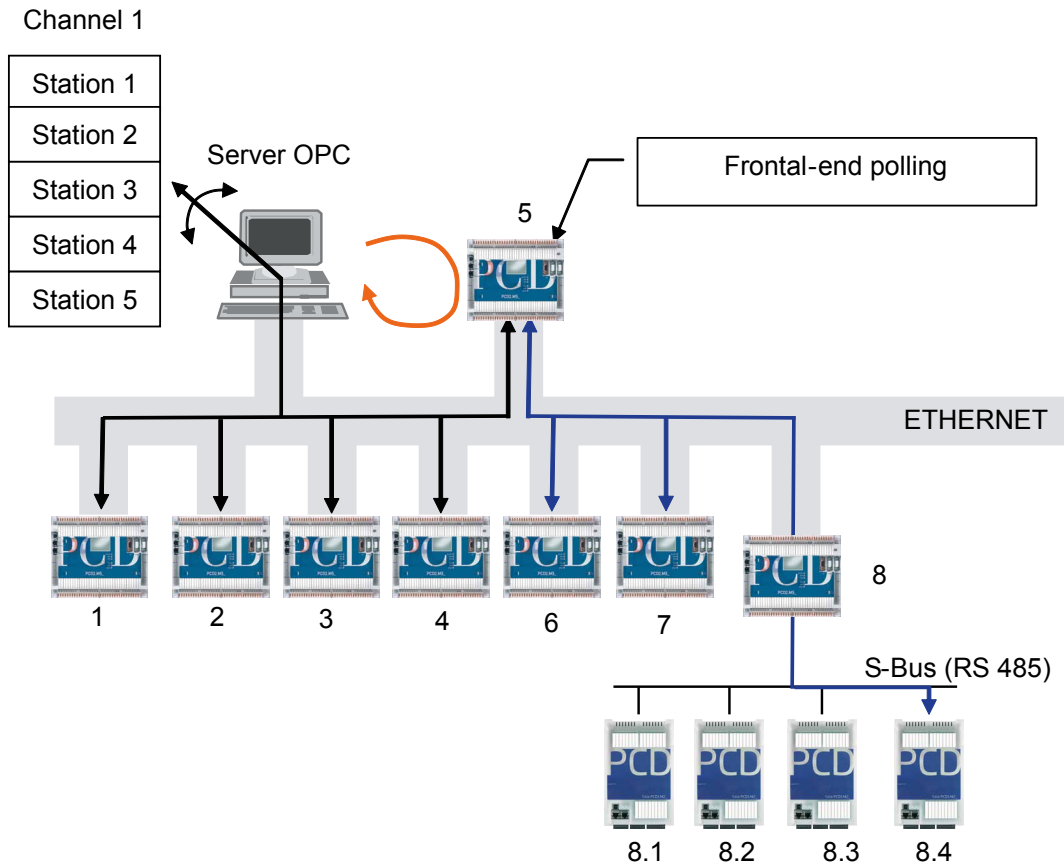
The telegrams are handled sequentially.

In order to communicate with S-Bus stations under a gateway station, it is therefore required that a single-channel connection be used for each OPC server or SCADA system.

Telegrams may be lost on a PCD2 gateway PCD if more than four clients are simultaneously accessing PCD stations connected under the gateway station. When using a PCD1 as a gateway, this restriction applies for even a single client.

**3.2.6 Rules for OPC servers**

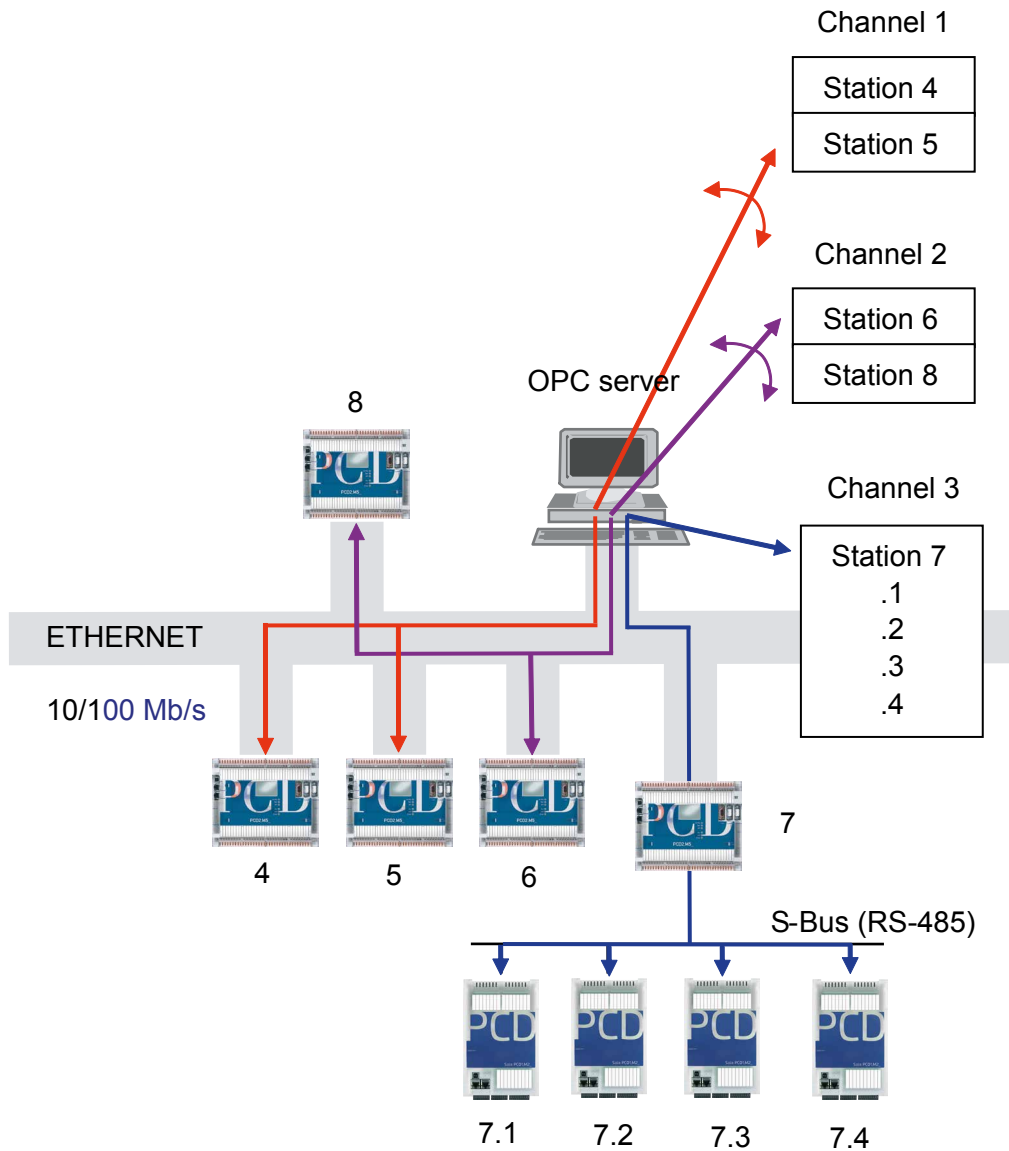
**Standard solution using one channel**



The standard solution is to process multiple PCD stations with a single channel on the OPC server. PCD stations 1, 2, 3, 4, and 5 are connected one after the other in series.

It is also possible to poll a "front-end PCD" which receives data from several other PCD stations, e.g. 6, 7 and 8.

**Solution using multiple channels**



If the standard solution is not possible because the PCD stations need to be processed faster, several channels can be set up in parallel on the OPC server. This makes it possible to process the data from the PCD stations more rapidly.

Each channel represents one program task on the OPC server. The more tasks there are, the higher the load on the CPU.

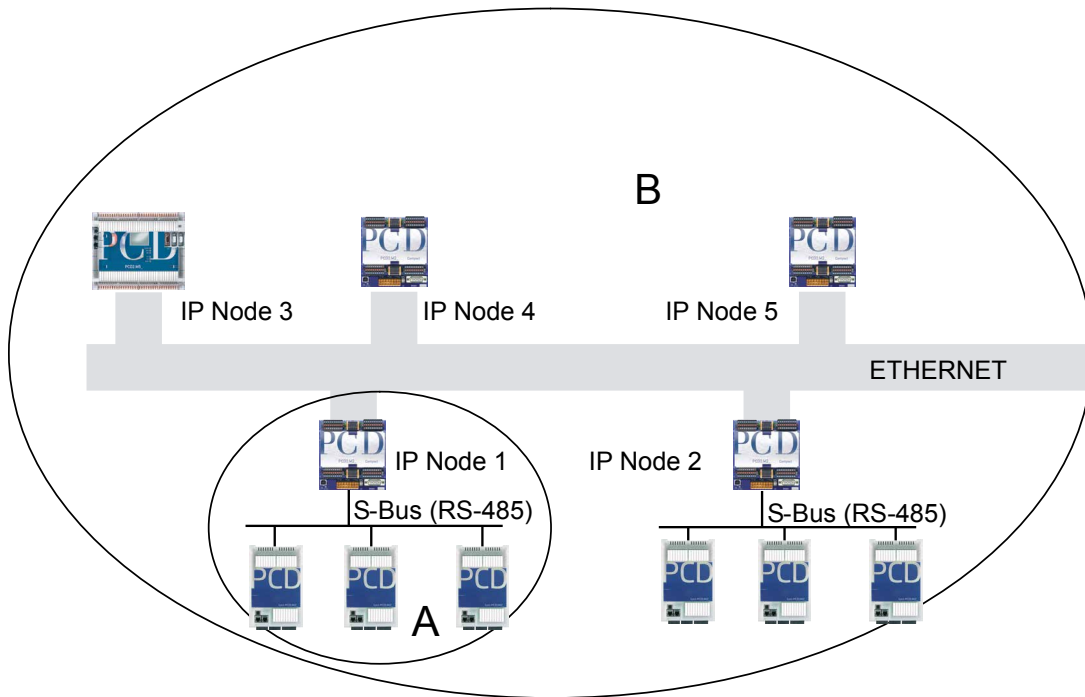


PCD1 stations on an S-Bus network under a gateway station can only be connected to the OPC server via a single channel, because the capacity of the buffer is only one telegram.

### 3.2.7 Broadcast telegrams

Broadcast telegrams for synchronisation with a client station can be transmitted in two different ways.

- A Restricted to the S-Bus network
- B To the whole Ethernet, including all S-Bus stations which are addressed under a gateway station.



IP node 65,535 (0xFFFF) is reserved for the transmission of broadcast telegrams via IP. It can be addressed in the IL and using FUPLA FBoxes.

<b>A</b>	Broadcast restricted to the S-Bus network	Send to IP node	X<65,535
		S-Bus	255
<b>B</b>	Broadcast on the whole Ethernet, including the S-Bus network under the gateway station	Send to IP node	65'535
		S-Bus	255
<b>C</b>	This type of broadcast is not permitted	Send to IP node	65'535
		S-Bus	X<255



**Client side:**

If type-C broadcast telegrams are transmitted, the client-station diagnostics will issue error messages. The telegram will not be sent by the client station.

- The NEXE flag is set.
- Flags 29 and 30 are set in the send diagnostic of the diagnostic register.
- The error LED illuminates.

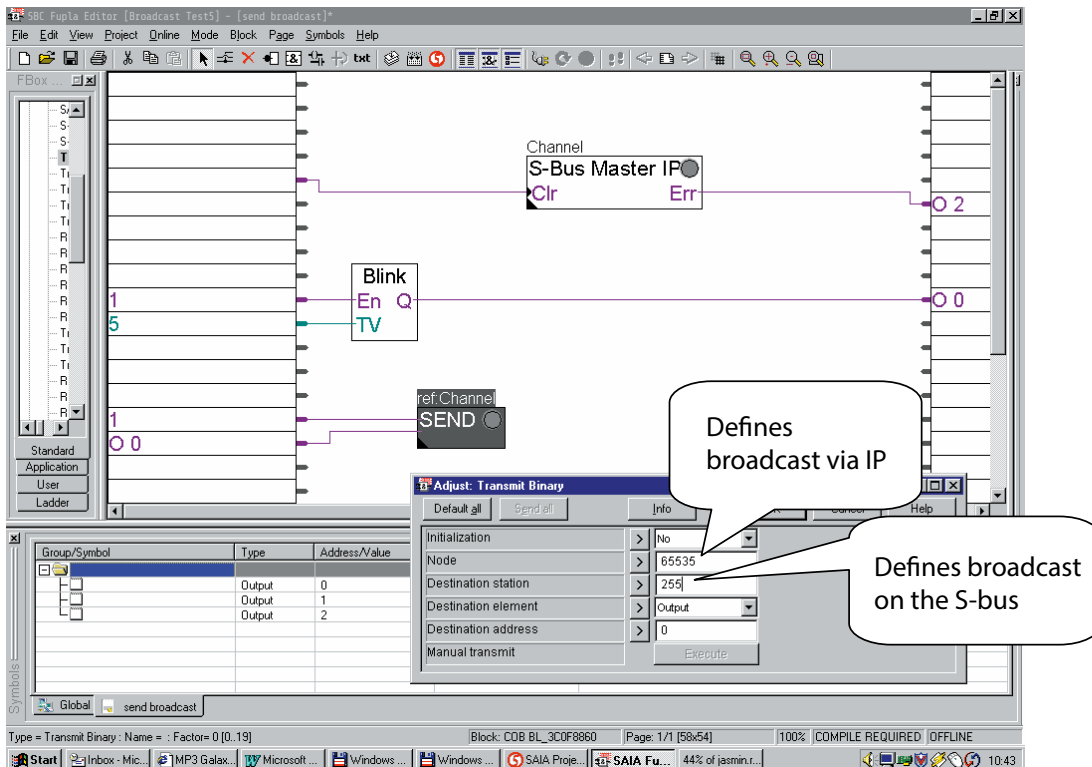
Furthermore, the Saia PG5® will not allow type-C broadcast telegrams to be sent.

**Server side:**

On the server side, type-A and B broadcast telegrams are processed without a response.

The server is unable to differentiate between standard S-Bus telegrams and type-C broadcast telegrams. For this reason, the PCD server station responds to this type of telegram.

See the following example, in which "Send Binary" is transmitted as a type-B broadcast telegram.



### 3.3 Open Data Mode via TCP/IP or UDP/IP

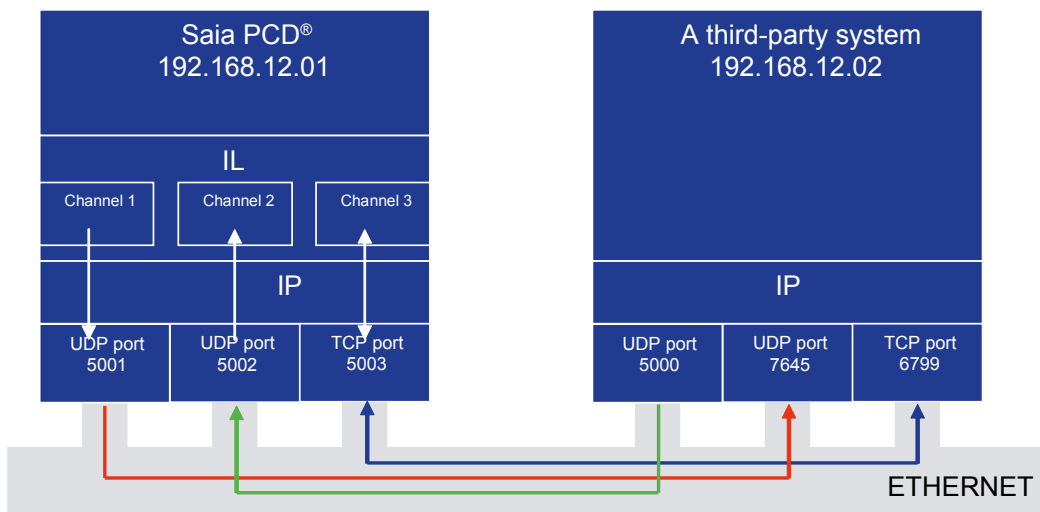
Using Open Data Mode makes it possible for a PCD to communicate with a station from a third-party manufacturer that does not support the S-Bus protocol. However, if necessary, two PCDs can also communicate with each other in Open Data Mode.

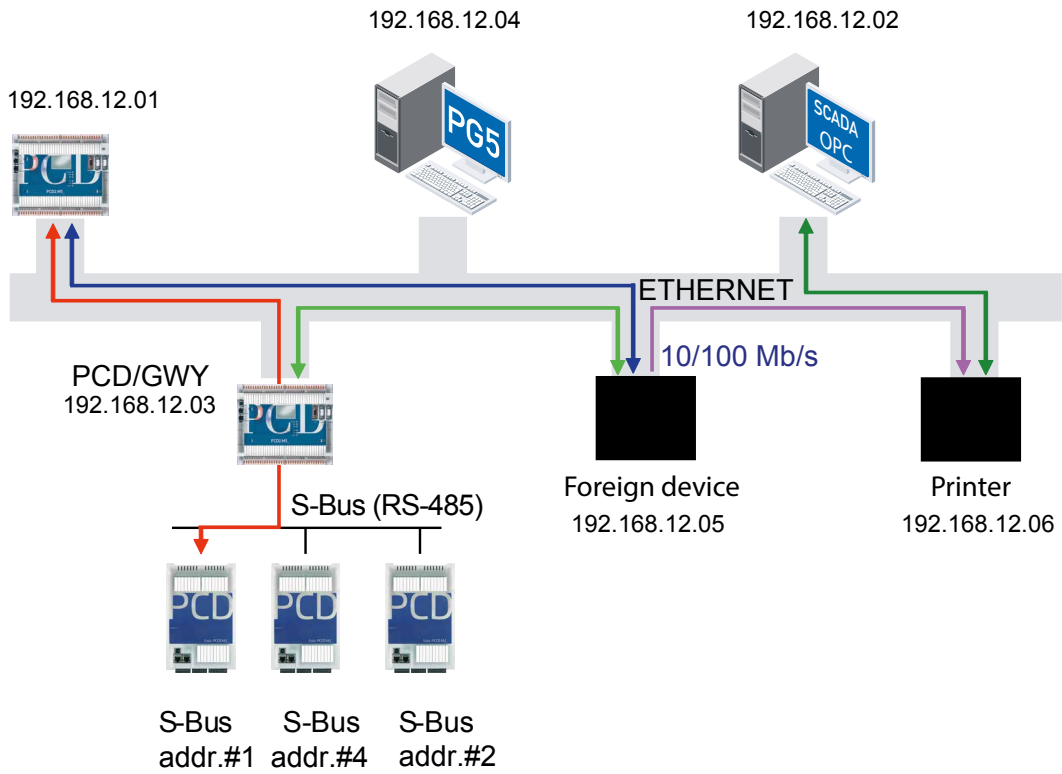
Stations from third-party manufacturers (e.g. printers, other PLCs, etc.) do not support the S-Bus protocol; therefore only unprocessed data (numbers, strings) can be exchanged. A PCD can send unprocessed, transparent TCP/IP or UDP/IP data packets. Implementation of the protocol is handled by the user application, practically without limits or restrictions. The implementation rules of the application are left entirely to the user. Possible options are multi-master communication, event-driven communication, etc.

The user can choose either TCP/IP or UDP/IP communication. The UDP variant is connectionless and the TCP variant connection-oriented. Under TCP/IP, a distinction is made between client and server when a connection is established.

In this type of protocol, the programmer uses the SBC API socket for direct access to the UDP or TCP/IP socket layer. A communication channel on the PCD is defined using the device's IP address and the communication port (UDP port / TCP port).

3





### 3.4 Layout and structure of the network

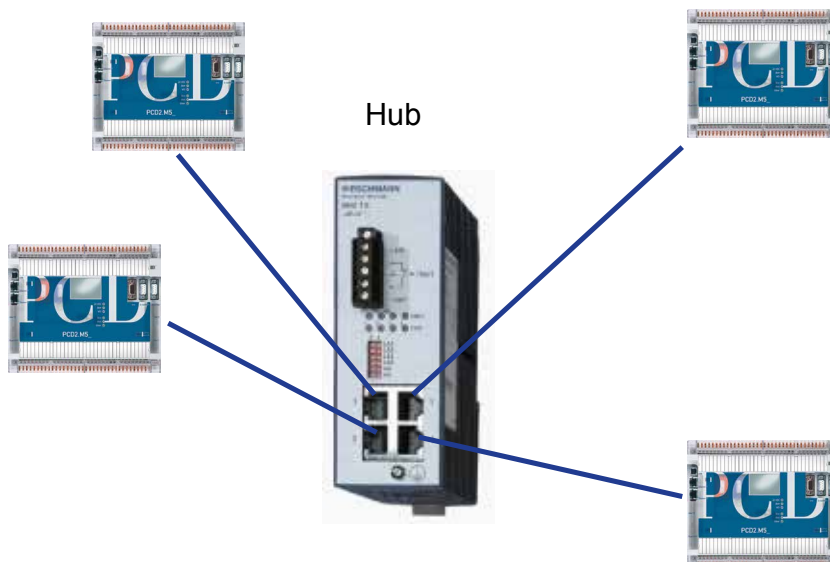
In addition to the PCD7.F65x TCP/IP Ethernet module, other components are also required to create Ethernet networks.

#### 3.4.1 Hubs (star network)

The layout of a hub network is similar to a star - a central hub with cables leading, like rays of a A hub merely transfers the signal that it receives on a data line. There is no logical connection. As opposed to a bus or ring network, this star topology reduces the vulnerability of the whole network to a single interruption in a cable. If the cable connecting an individual host to the network is damaged or disconnected, access to only this host is lost.

Reasons to use hubs:

- Low-cost connection to the units
- Short delays when forwarding data
- In a hub, the Ethernet analyzer can be used to view all data traffic. This is different from a switch, which handles communication in



#### 3.4.2 Switches

Switches do not simply transfer or repeat the signals transmitted on an individual cable; they receive all signals, process them, and then forward them. The result is that only valid Ethernet frames are transmitted to the other network. This filtering process occurs in the Ethernet layer by comparing the MAC address. The routing decision is made only once for each IP address, so each frame with the same IP address is forwarded accordingly.

Local data traffic remains local. This minimises the number of data collisions and optimises network usage. This creates collision-free areas.

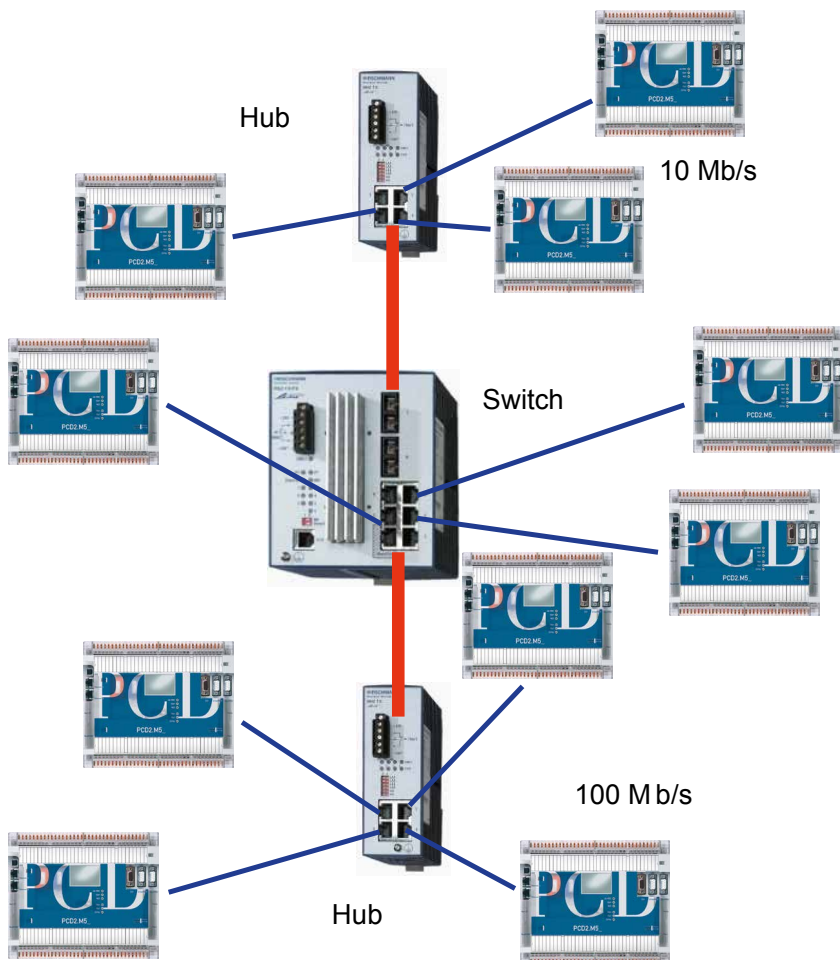
Two Ethernets can be connected with switches, which transfer traffic from one network to the other.

One switch can process several telegrams simultaneously.

Reasons to use switches:

- Creation of collision-free areas: improves the deterministic behaviour of the network due to the low number of collisions
- Combination of different connection types (10/100 Mbit/s, half-duplex HDX / full-duplex FDX)
- More efficient flow of data because of point-to-point connections and full-duplex operation
- Improved network performance due to filters (broadcast and multicast filters) and prioritisation procedures.

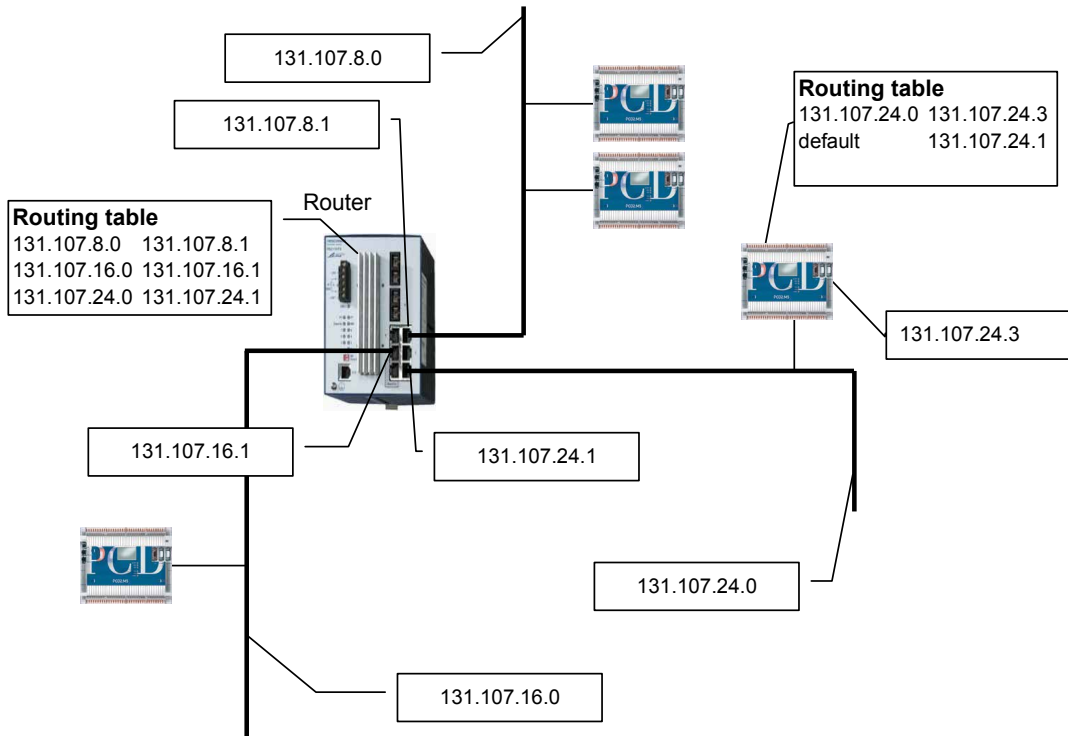
3



### 3.4.3 Router

A router is a peripheral interface device which transfers telegrams from one physical network to another. The sending host and the router must select a transfer destination for the telegram. This routing selection occurs when the IP stack consults the routing table. The routing table contains the IP address entries and identifies the network interfaces for the router. By default, a router sends telegrams only to those networks for which it possesses a configured interface.

- If a host would like to communicate with another host, the IP protocol must first decide whether the destination host is located on the local network or on a remote network
- If the destination host is located on a remote network, the IP protocol searches in its routing table for a route to the remote network
- If no obvious route is located, the router's default address is used.
- The routing table on the router is consulted so the telegram can be forwarded to the specified network.



### 3.4.4 Network components

Saia Burgess Controls does not supply any components specifically for Ethernet networks. We have had good experience with network components from the Ethernet Rail family produced by Hirschmann.

The industrial line ETHERNET Rail family was developed specifically for use in industrial automation applications. It supports redundancy functions (alternative data path via HIPER-Ring structures, redundant connection of network segments), and guarantees high levels of network availability.

Hirschmann also provides support in developing networks for customer projects:  
[www.hirschmann.com](http://www.hirschmann.com)

### 3.4.5 Increasing performance

#### Full-duplex, auto-negotiation, auto-sensing

The PCD7.F65x Ethernet TCP/IP module supports "full-duplex" and "half-duplex" modes. Duplex is not a network topology, but rather a method for exchanging data between two nodes.

The PCD7.F65x Ethernet TCP/IP module uses auto-negotiation and auto-sensing to establish a compatible mode between two nodes. As soon as two nodes are connected, the preferred data communication mode is set according to the following list:

- 100Base TX and full-duplex mode
- 100Base TX and half-duplex mode
- 10Base T and full-duplex mode
- 10Base T and half-duplex mode

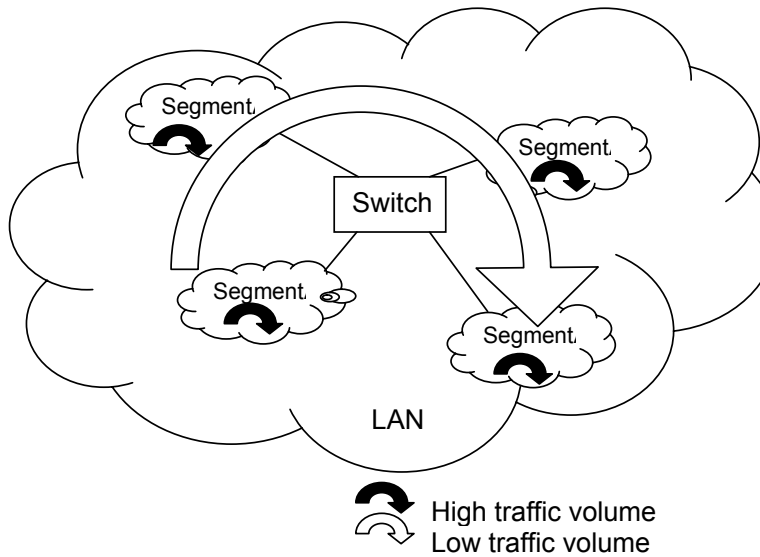
If auto-negotiation from the PCD to the partner station is not possible, 10Base T and half-duplex mode is implemented.

Auto-sensing is a feature which allows the data rate of a signal (10 Mbit/s or 100 Mbit/s) to be detected. Auto-sensing is possible even without auto-negotiation.

#### Switched LAN compared to shared LAN

It is also possible to divide the network into different groups to achieve a separate Ethernet without putting unnecessary strain on overall network traffic. For this reason, standard switch technology should be used in order to create collision-free areas.

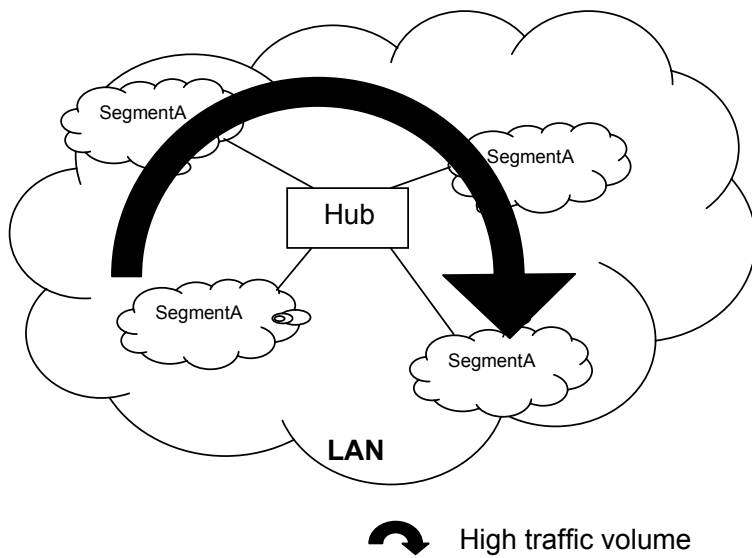
**Switches LAN**



- Each segment has full performance/data rate
- Simultaneous data traffic in multiple segments
- Local data traffic remains local. Only selected telegrams exit the local segment.

3

**Shared LAN**



- All segments share performance/data rate
- All telegrams pass through all segments
- There is only one telegram at a time in the network
- Collisions reduce efficiency to 40%

**Maximising user data (payload)**

Note that both TCP/IP and UDP/IP (particularly TCP/IP) have a quite large overhead. This overhead is of particular importance for telegrams containing little content. To benefit fully from the performance of the Ethernet, combining large quantities (i.e. 32 registers on the S-Bus via UDP/IP) of user data for transmission is recommended.

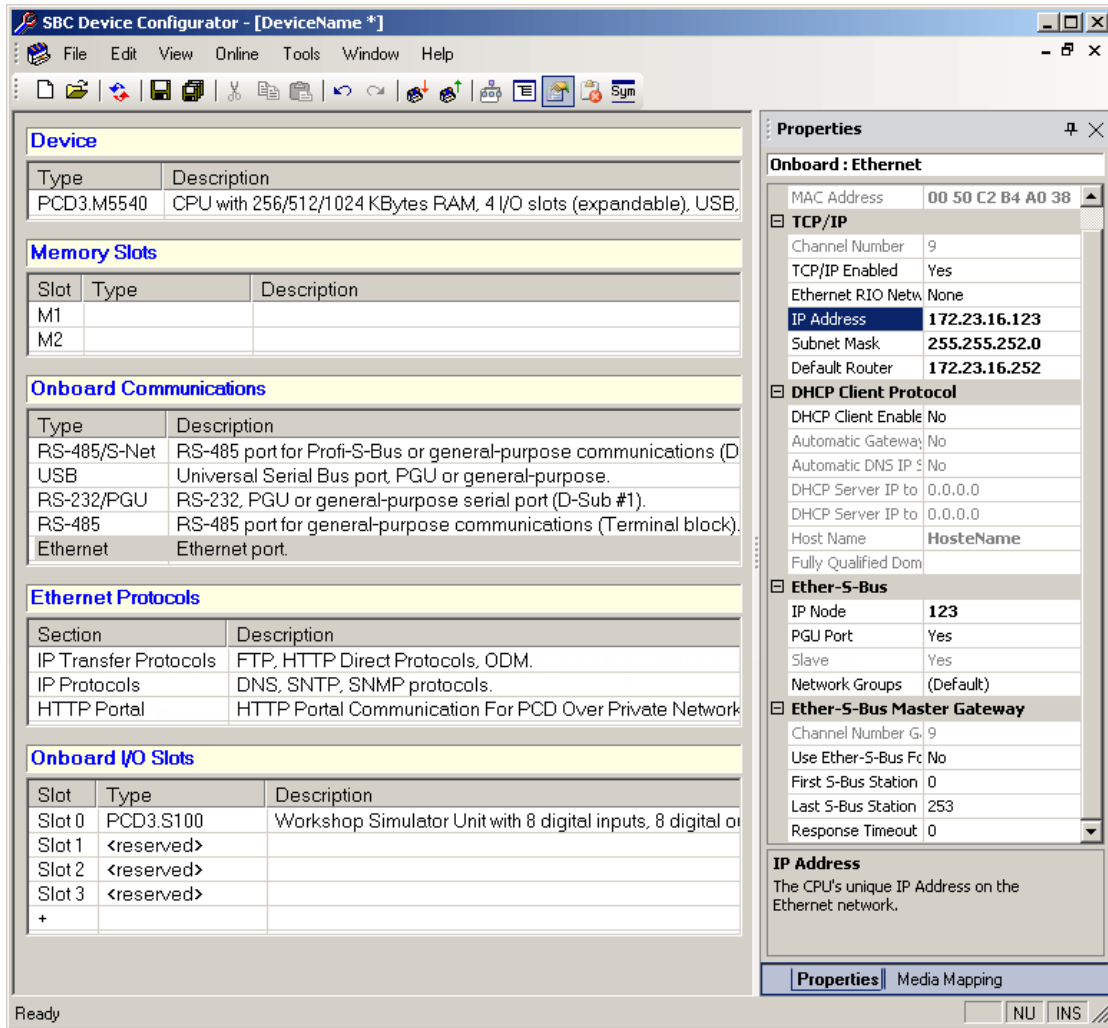


## 4 Configuration and programming

### 4.1 Configuration and addressing

#### 4.1.1 Configuring the S-Bus IP port (server)

The IP module is configured in Saia PG5® → Device Configurator:

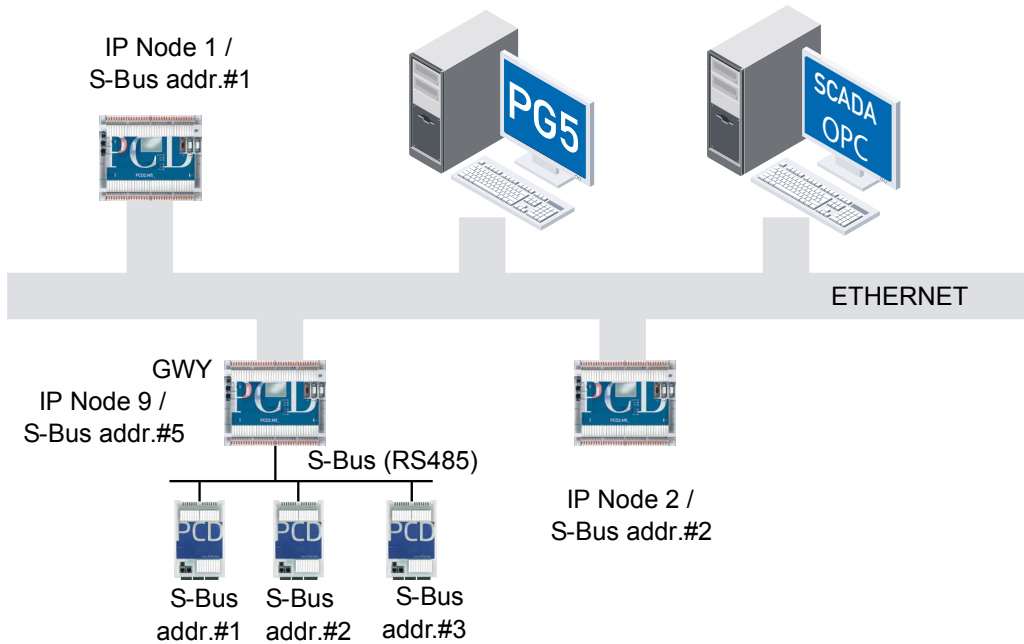


**4.1.2 Addressing the IP server station**

In principle, each IP module is defined by an Ethernet (MAC) address unique in the world and a location-dependent IP address. The conversion (MAC address ↔ IP address) is automatically performed by the ARP (Address Resolution Protocol) in the TCP/IP stack.

PG5 generates a IP Settings table (DBX) to connect the IP addresses, IP nodes, and S-Bus addresses with all the CPU hardware information for the project. When programming, the user requires only the S-Bus addresses and IP nodes. The DBX is generated when the project is created, and loaded into the PCD via the user program. If an IP address, S-Bus address, or IP node in the project is changed, the programs of all concerned CPUs must be rebuilt (generation of a new DBx) and loaded into the PCD. You can find more information in the PG5 help file.

Word-encoded addressing for IP nodes and a byte-encoded S-Bus address offer suitable addressing options for the Ethernet. They enable 65,536 IP nodes or 65,536 x 253 PCDs (253 PCDs on an S-Bus network under an Ethernet S-Bus gateway, see figure) to be addressed. PG5 generates a DBx showing the relationship between IP nodes and IP addresses.



For stations on an S-Bus network under a gateway in IP node #9, S-Bus address #5 should no longer be used, as it is already being used by the gateway.

Conversion table (IP address - IP node - S-Bus address)

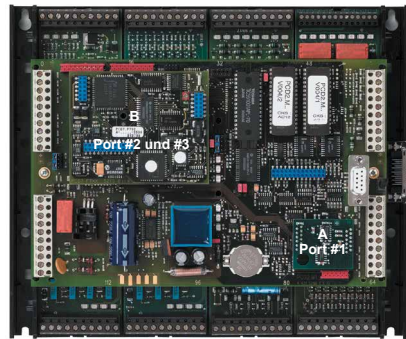
IP-adress ↔...	...IP node	...S-Bus address
192.168.2.151	2	2
192.168.2.155	3	9
192.168.2.168	6	--
192.168.2.201	7	--
192.168.2.105	9	1-254, excl. 5

### 4.1.3 Slots and channel numbers

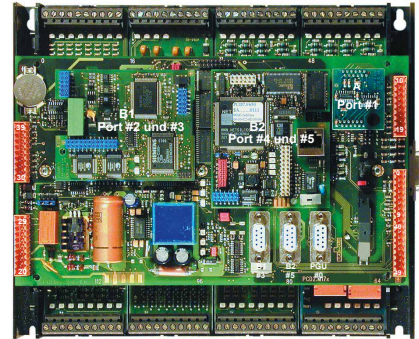
The slots/channels for the PCD7.F65x are defined on the Saia PCD® as follows (see also Chapter 2):



PCD1.M130  
Channel 9  
Slot B

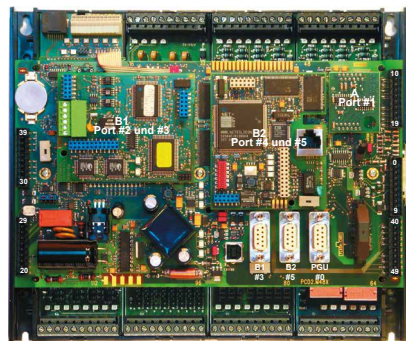


PCD2.M15x  
Channel 9  
Slot B



PCD2.M17x  
Channel 8  
Slot B2

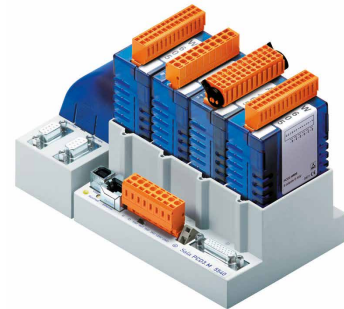
4



PCD1.M480  
Channel 9  
Slot B1  
and also  
Channel 8



PCD4.M17x  
Channel 9  
Slot B1



without PCD7.F65x  
Ethernet-capable  
PCD3.M5540  
PCD3.M3120  
PCD3.M3330  
PCD3.M6340  
PCD3.M6540  
Channel 9

## 4.2 Programming the S-Bus via Ethernet

The Ethernet S-Bus protocol is used for communication between two PCDs, one PCD and a programming tool (PGX), or with a third-party unit (PLS with scom.dll or PLC with S-Bus support).

The new master/master functionality on a network is an improvement over the standard S-Bus protocol. On the standard S-Bus, only one master is allowed per network. With S-Bus via Ethernet, all stations can act as masters.

This type of communication is controlled by the well-known STXM/SRXM commands. The syntax is similar to standard S-Bus telegrams.

Attributes	
Procedure	Command request Request (client) • Indication (server) • Response (server) • Confirmation (client)
Communication security	Maximum of 3 repeated attempts in the background. S-Bus CRC 16 error checking is used. No special secure layer is used via IP.
Protocol via IP.	The UDP protocol is used for communication on the S-Bus via Ethernet. The receive communications-socket is open and connected permanently to port 5050. The send communications-socket can be connected either to port 5050 (default) or to another free port in the system.

### 4.2.1 Description of Saia PCD® commands

The IP supports the following instructions:

SASI(I)	Assign serial interface
STXM(X)	Serial send medium for data
SRXM(X)	Serial receive medium for data

#### SASI initialisation

In addition to the configuration in the offline hardware configurator, a SASI for client and slave has been provided. In the interests of backwards compatibility, the number of SASI parameters remains unchanged. As S-Bus via Ethernet uses a fixed, predefined input communication port (5050), there is no need to define a special communication port. An unused port between 1024 and 4999 is selected as the output communication port. However, as described below, the output port can also be connected to port 5050 in the SASI text.

Usage: SASI Channel ;slot (8 or 9)  
Text no. ;definition text no. 0 - 7,999

Example: SASI 8 ;initialise slot 8  
IP text ;definition text for SASI

Defintion of SASI text:

Master	"<mode_def>,<dest_reg>; <diag_def>; <option_def>"
Example	"MODE:EM,R100;; DIAG:F1000,R1000; TOUT:500,PORT:5050"
Slave	"<mode_def>,<dest_reg>; <diag_def>;"
Example	"MODE:ES;; DIAG:F1000,R1000;"

"<mode\_def>,<dest\_reg>; defines the mode

In Ethernet master mode, the address of the remote station is appended to the EM.

Master	"MODE:EM,<dest_reg>;"	EM: Ethernet master mode. <dest_reg>: The register defines the address of the remote station.
Slave	"MODE:ES;"	ES: Ethernet Slave mode (only used for diagnostics on the slave). Note: the Ethernet Slave is configured automatically when the TCP/IP module is configured in the PG5 "Hardware Settings".

<dest\_reg>: represents the register which defines the partner station. (type: Rxxxx)

Before STXM/SRXM commands can be sent, the destination address must be stored in an address register, as is also the case with standard S-Bus. This addressing scheme uses two address fields.

Register address field (32 bit)		
MS word		LS word
IP node number of the destination PCD	Not used	S-Bus address of the destination slave

<diag\_def>; defines the diagnostic elements for Ethernet communication.

Format: "DIAG:<diag\_flag>,<diag\_reg>;"

Type		Description
<diag_flag>	Fxxxx Oxxxx	Base address of 8 consecutive flags or outputs
<diag_flag>	Rxxxx	Base address of the diagnostic register

<option\_def>" Defines option elements for Ethernet communication.

Format: "[<tout\_def>],[<port\_def>;]"

Type		Description
<tout_def>	TOUT: xxx	Represents the timeout value for EM mode in ms. The default timeout value is 500 ms. (Lower limit value = 100 ms)
<port_def>	PORT: xxx	Selection of the output port for S-Bus UDP communication Default: xxx = 0 0 = a free port between 1024 and 4999 is selected xxx = 5000 to 65,535 implies fixed assignment by the user

**Flags:** If the definition text is missing or invalid or if the firmware does not offer IP support, the error (E) flag is set.

## Diagnostics

### Diagnostic flags

Address	Name	Description
xxxx	RBSY	Receiver busy
xxxx+1	RFUL	Receiver buffer full
xxxx+2	RDIA	Receiver diagnostic
xxxx+3	TBSY	Transmitter busy
xxxx+4	TFUL	Transmitter full
xxxx+5	TDIA	Transmitter diagnostic
xxxx+6	XBSY	Physical link
xxxx+7	NEXE	Not executed

**Transmitter Busy (TBSY) ↑** Indicates that transmission is in progress.

Significance formaster station: If an STXM or SRXM instruction is set to “H” (high) during execution, the flag is reset as soon as a valid response is received.

Slave station: Is set to “H” while the response is being sent.

**Transmitter Diagnostic (TDIA) ↑** Indicates that an error was detected during transmission of a telegram. A detailed description of the error can be found in the diagnostic register (bits 16 to 31). The flag is reset as soon as all sender diagnostic bits (16 to 31) have been reset in the diagnostic register.

**Not Executed (NEXE) ↑** Indicates that an instruction (STXM or SRXM) has not been executed after three before the timeout. The flag is reset by the following S-Bus instruction.

**Physical Link (XBSY) ↑** Indicates whether or not there is a physical link. Set to “H” (high) whenever the Ethernet TCP/IP module detects a physical link and to “L”(low) when the physical link is lost. As soon as the SASI (master or slave) command is executed, this flag is assigned and can be used as a diagnostic. If the PCD sends telegrams without a physical link having been established, bit 23 is set in the diagnostic register.

## Diagnostic register

	Bit	Designation	Description
RECEIVER	0	Overflow error	Overflow in the internal receive buffer
	1	Not used	
	2	Framing error	Usually caused by an incorrect baud rate
	3	Break error	Break in the data line *)
	4	BCC error	Bad Block Check Code or CRC-16
	5	Not used	
	6	Not used	
	7	Not used	
	8	Length error	The telegram has an invalid length
	9	Not used	
	10	Inv tlg	Invalid or illegal telegram
	11	Not used	
	12	Range error	Invalid element address
	13	Value error	Error in the received value
	14	Missing Media	Media invalid or does not exist
TRANSMITTER	15	Not used	
	16	Retry count	Shows the number (binary) of repeated attempts. (telegram repetitions in binary form)
	17		
	18		
	19		
	20	NAK response	Negative acknowledgement (NAK) received
	21	Missing response	No acknowledgement received before timeout
	22	Inv. response	ACK/NAK received instead of data, or data instead of ACK/NAK
	23	Target not present	Target station does not exist or physical link has been lost
	24	Not used	
	25	Not used	
	26	Not used	
	27	Not used	
	28	Range error	Invalid element address
29	TxNode_error	Node does not exist	
30	TxBroadcast_error	Error sending a broadcast telegram via IP	
31	Program error	Impermissible send attempt	

\*) Has no significance for SM0/SS0 mode

Each bit which is set to "H" (high) in the diagnostic register remains in this state until it is manually reset by the user program or the debugger.



<b>Overrun Error (bit 0)</b>	Set to "H" when an overrun occurs in the internal DUART buffer.
Cause: →	The assigned transfer rate is too high. The CPU is no longer able to process all the figures it receives. This may occur when a CPU is engaged in communication connections requiring a high data transfer rate across several interfaces at the same time. It is theoretically possible for all interfaces of a CPU (with the exception of the 20 mA current loop) to be assigned a maximum transfer rate of 19,200 bit/s. However, in practice, this error can occur when there is a high volume of communication traffic. The system program assigns different priorities to the interfaces. The highest priority is assigned to interface 0, decreasing in importance down to interface 3.
Remedy:	<ul style="list-style-type: none"> <li>• Reduce</li> <li>• Where possible, use an interface with a higher priority for rapid communication.</li> </ul>
<b>Framing Error (bit 2)</b>	Set when a figure with a framing error is received (missing stop bit). This is usually caused by the wrong transfer rate being set.
<b>Break Error (bit 3)</b>	Set to "H" when an interruption occurs in a figure being received.
Cause:	Data line broken or incorrect transfer rate set.
<b>BCC or CRC-16 Error (bit 4)</b>	Set to "H" if a CRC-16 error is detected in the incoming telegram. The incoming telegram is rejected.
Reaction of the slave:	The received telegram is ignored.
Reaction of the master:	The received telegram is ignored and the last telegram is resent.
Cause:	Malfunction on the data line.
Remedy:	Check the electrical installation
<b>Length Error (bit 8)</b>	Set to "H" if a telegram is received which has an invalid length. The error cannot occur on a network consisting exclusively of PCD stations (no third-party stations). The error indicates that an invalid telegram has been received from an external system. This leads to a NAK response.
<b>Address Error (bit 10)</b>	Set to "H" if an invalid telegram is received (wrong command code).
Cause:	The same as for a Length Error (there is no NAK response).
<b>Range Error (bit 12)</b>	Set to "H" if an incoming telegram contains an invalid PCD element address. This error cannot occur on a network consisting exclusively of PCD stations, as the master PCD monitors the range of the element address when it is sent. The slave station responds to this error with a NAK.



<b>Value Error (bit 13)</b>	Set to “H” when an invalid data value is received.
Example:	The STXM instruction is used to load the time. The value received for the hour is 30. However, the permissible range for the hour is only 0...23. The slave station responds to this error with an NAK.
<b>RxBroadcast Error (bit 14)</b>	Set if an invalid broadcast telegram is received (IP broadcast → IP node = 65,535 and S-Bus address < 255).
<b>Retry Count (bits 16 to 19)</b>	Shows in binary form the number of repeated telegrams sent during the execution of an SRXM or STXM instruction . Bit 16 represents the LS bit (bit with the lowest value). The quality of an S-Bus network can be judged by monitoring these two bits.
<b>Negative Response (bit 20)</b>	Set to “H” if an NAK response is received from a slave. This means that the master previously sent an invalid telegram. Check for the following errors: Value Error, Range Error, and Length Error.
<b>Missing Response (bit 21)</b>	Set to “H” if no response has been received by the slave station before the timeout. In such a case, the telegram is resent (a maximum of twice).
Possible causes:	The addressed slave station does not exist. Network installation error (wiring). The slave station has received an unintelligible telegram with a CRC-16 error.
Remedies:	Check the slave station (connections, stationnumber). Have the right cable connectors and pull-up/down resistors been connected to the bus line at the first and last station?
<b>Multiple NAK (bit 22)</b>	Set to “H” if another response is received from a slave station, instead of the anticipated ACK or NAK.
Possible causes:	<ul style="list-style-type: none"> <li>● More than one slave is using the same stationnumber.</li> <li>● There is more than one master on the network.</li> <li>● Malfunction on the bus line.</li> </ul>
Remedies:	See solutions for a missing response error
<b>Target not present (bit 23)</b>	Set to “H” if the destination station cannot be contacted on the network. Either after three attempts or if there is no physical link.
Possible causes:	Defective connector cable or interruption of the destination station’s power supply

<b>Range Error (bit 28)</b>	Set to "H" if the SRXM or STXM instructions exhibit an element address (origin or target address) outside of the permitted range.
Possible cause:	Error in the user program Monitored ranges: Inputs/Outputs 0...8191 Flags 0...8191 Timer/Counter 0...1599 Register 0...4095
Example:	While the subsequent STXM instruction is being executed, the bit for Range Error is set to "H". STXM 1 ; channel 1 25 ; 25 Register R 1000 ; origin of the base address R 4072 ; destination of the base address An attempt is made to send the contents of registers 1000 to 1024 in the master station to registers 4072 to 4096 in the slave station.
<b>TxNode Error (bit 29)</b>	Set to "H" if the node does not appear in the node list, if it is not configured, or if an invalid broadcast telegram has been sent (IP broadcast → IP node = 65,535 and S-Bus address < 255).
<b>TxBroadcast Error (bit 30)</b>	Set to "H" if an invalid broadcast telegram is sent (IP broadcast → IP node = 65,535 and S-Bus address < 255).
<b>Program Error (bit 31)</b>	Set to "H" if the interface was assigned in SS1 mode during the execution of an STXM or SRXM instruction or if a similar instruction has already been executed (TBSY flag was not polled before the instruction was executed).

### Sending/receiving STXM/SRXM data to/from the slave station

The instructions are the same as those for the S-Bus. The sole difference is the channel number (8 or 9) and the IP node address of the partner station.



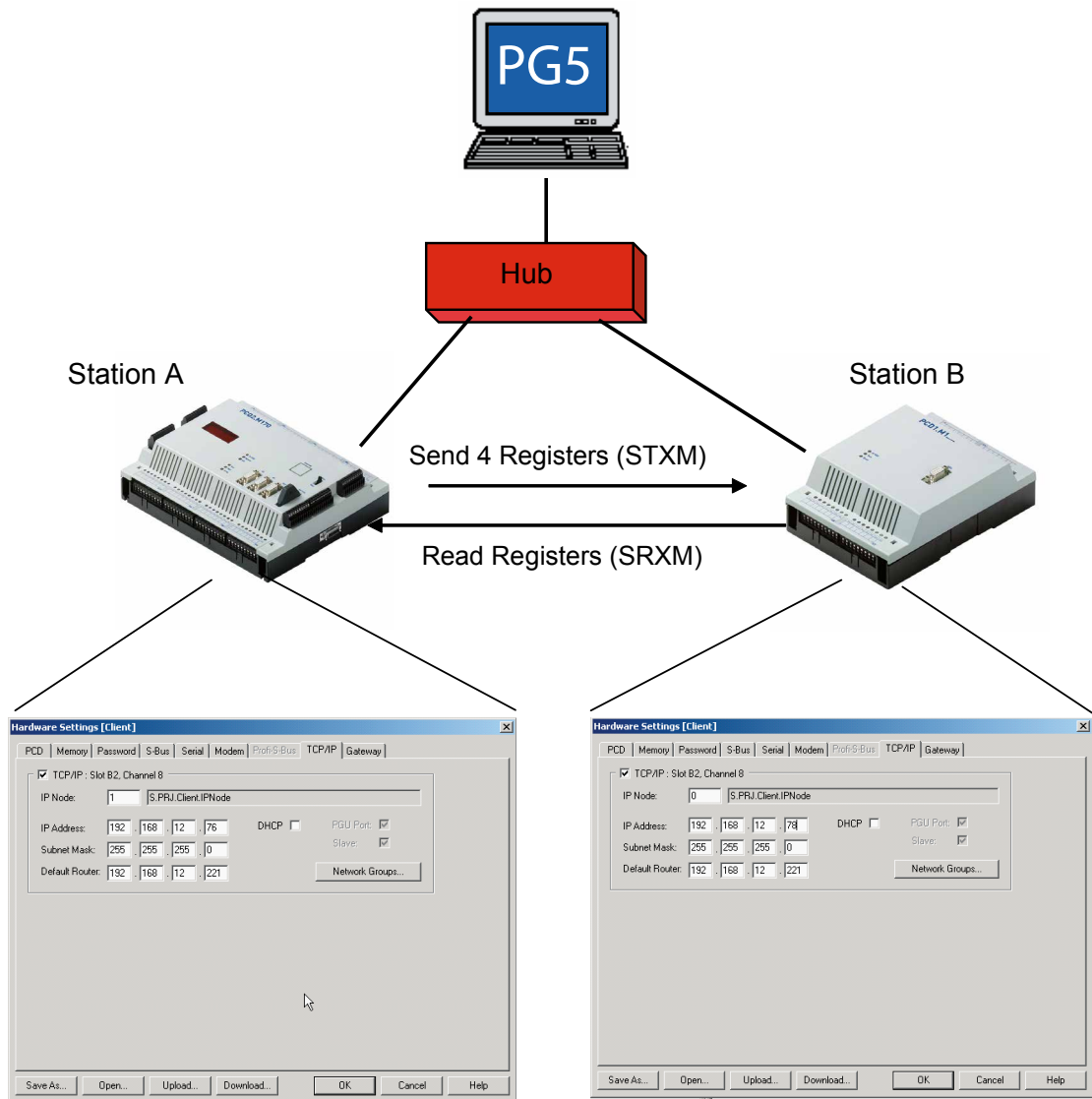
More details can be found in the S-Bus manual.

**Example of use:**

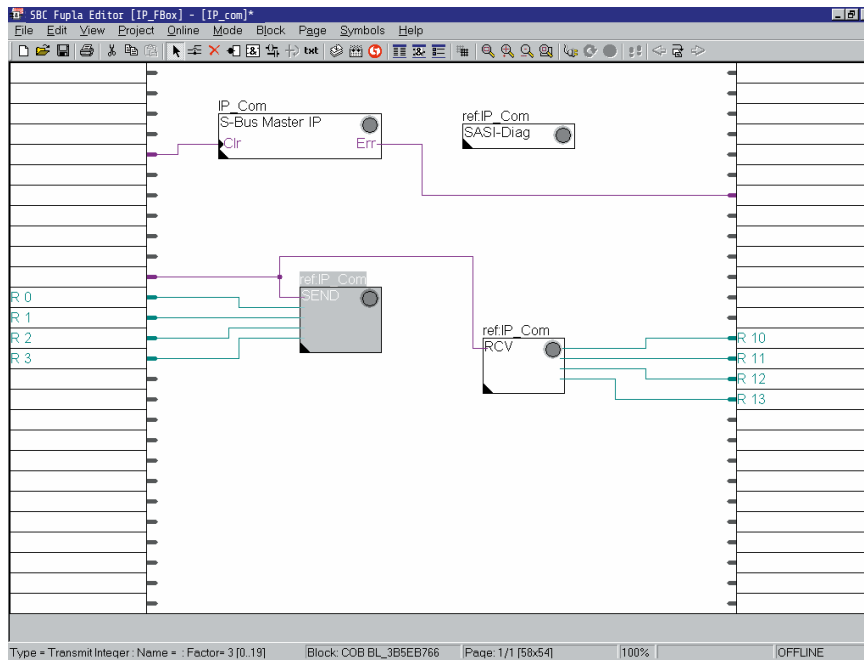
```
LDL R 100 ; destination address register
      3 ; S-Bus address 3
LDH R 100
      15 ; IP node 15
STH TBSY ; check busy flag for sending
JR H
next
STXM 9 ; slot 9 / B1
      4 ; four elements
      F 500 ; origin address
      O 32 ; destination address
next:
```

**4.2.2 Using Saia PG5® FBoxes to program the Ethernet S-Bus**

**Example:**



Station A sends/receives 4 registers to/from station B practically simultaneously.

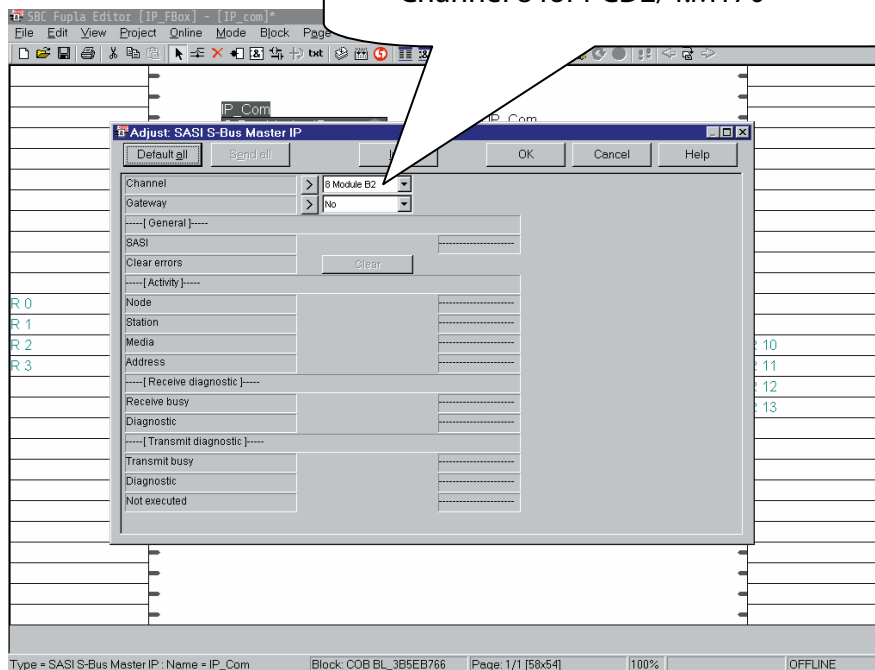


4

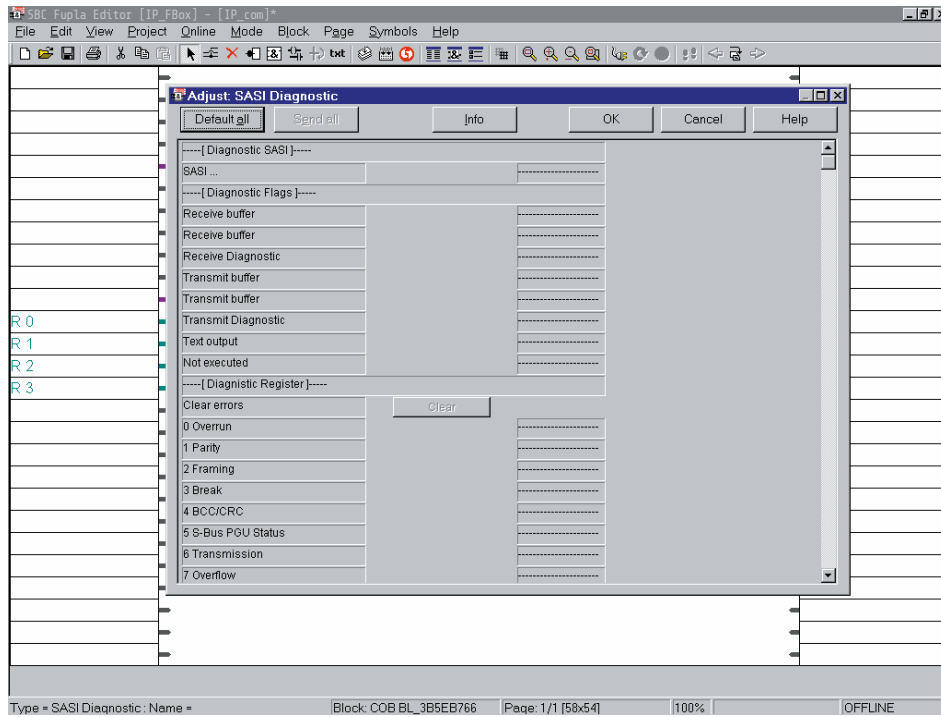
### SASI IP master setting

Channel to select in the SASI setting:

- Channel 9 for PCD1.M130, PCD2.M150
- Channel 8 for PCD2/4.M170



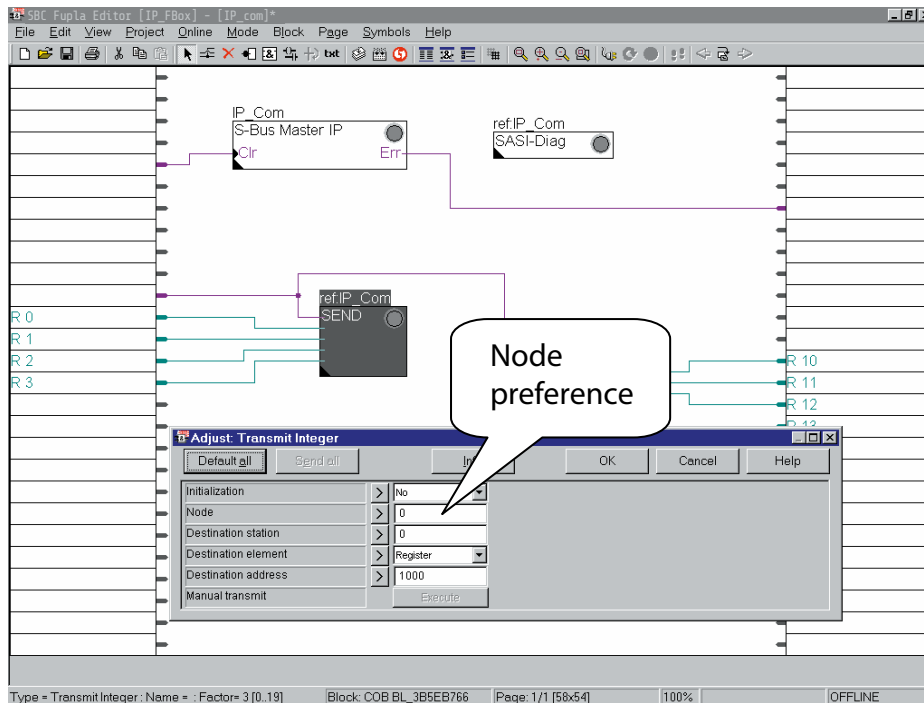
### SASI IP diagnostic setting



4

### Send/Receive setting

The sole difference between the IP S-Bus and the serial S-Bus is the node in the addressing setting.



### 4.2.3 S-Bus IP multimaster

Several masters can be connected to an S-Bus IP slave port. Each master gateway (MGWY) has 4 request buffers (PCD1, 1 buffer), which are reserved for processing 4 parallel requests to the MGWY.



You can find more details in Chapter 3 “Attributes and functions”.

### 4.2.4 Broadcast telegrams via S-Bus IP

Two different transmission types are allowed for broadcast telegrams (for synchronisation with a client station):

- Broadcast restricted to the S-Bus network (without Ethernet)
- Broadcast over the whole Ethernet, including all S-Bus stations under the gateway station addressed.



You can find more details in Chapter 3 “Attributes and functions”.

### 4.3 Programming the Open Data Mode via Ethernet

Open Data Mode is generally used for PCD communication with a station from a third-party manufacturer which does not support S-Bus. This makes it possible to implement any third-party protocol. If necessary, two PCDs can also communicate using the Open Data Mode.

Stations from third-party manufacturers do not support proprietary protocols (such as S-Bus). For this reason, only unprocessed blocks of data (numbers, strings without headers) should be transferred via IP.

While the PCD can send data to a remote station, in client mode, no data can be requested directly from the remote station. The data received in Open Data Mode is sent to the application layer.

When using the UDP transfer protocol, it is not possible to determine whether the connection between two stations has been broken. This feature must be implemented by the user in the application layer. A communication monitoring procedure able to detect interruptions to communication is included in the TCP protocol.

#### 4.3.1 Description of Open Data Mode

In Open Data Mode, the unprocessed data is appended to the UDP or TCP header and then sent. When using S-Bus via UDP, the data is always attached to the UDP header.



#### 4.3.2 Configuration

The IP module must be configured for S-Bus via IP with the IP address, subnet mask and default router, using the Hardware Settings in PG5. No further configuration is necessary. Open Data Mode is initialised by the command InitODM in the user program.

### 4.3.3 Important characteristics of UDP and TCP in Open Data Mode

#### General characteristic:

In Open Data Mode UDP and TCP, owing to the size of the mailbox between the Ethernet TCP/IP modul and the PCD, ensure that the maximum length of transmitted user data does not exceed 1536 bytes (720 bytes with PCD7.F650/F655) per telegram. Data exceeding this length will be lost during transmission.

#### TCP characteristics:

TCP is a connection- and flow-oriented protocol. Thus, it may be the case that individual parts of a telegram are sent over the Ethernet separately or that several telegrams are combined into one. It is the duty of the user at the receiving end to reassemble the telegrams again as needed. TCP guarantees, however, that the data will arrive in the correct order. TCP has implemented a check layer which guarantees the transmission of any packets that must be repeated. Using TCP on the Ethernet, no data will be lost.

TCP provides information about the status of the connection.

In TCP, the client/server principle is predominant. Before the two communication end points connect, they are designated as either the client or the server. The server awaits a connection from a client. A server is able to connect to different clients via the same input port. The client, however, is only able to connect to one server via its single output port. If a connection is established between the client and the server, both sides are able to send data. At this stage, both are effectively clients. Both are also able to terminate a connection.

#### UDP characteristics:

UDP is a connectionless, telegram-oriented protocol. Complete telegrams are sent individually, and only once. However, there is no layer which checks the data traffic, as exists for repeated packets with TCP. There is no guarantee that the telegrams will reach their destination. The user must program their own checking mechanisms, such as handshakes. Every station is able to send telegrams to every other station, meaning that there is no client/server relationship. Using UDP, it is also impossible to receive telegrams containing more than 1536 bytes (720 bytes with PCD7.F650/F655) of user data (the size of the communication mailbox).

A global diagnostic flag provides information as to whether there is a connection with the nearest hub/switch. This connection can be tested at any time, for example before a UDP telegram is sent.



#### 4.3.4 Programming with IL

Open Data Mode is programmed using system functions.

##### **Call System Function**

This is a type of instruction extension with various libraries. The call resembles an FB call. A library is provided for the Ethernet TCP/IP module. This library is part of the firmware. Functions are called using the CSF instruction. CSF instructions are employed to communicate between the Ethernet TCP/IP module and the PCD via a shared mailbox (send buffer, receive buffer). Thus the user of CSF instructions controls the communication of Open Data Mode and is responsible for the correct flow of communication via the mailbox. This includes, for example, ensuring that no telegrams are left in the mailbox.

Definition of the CSF (Call System Function):

```
CSF [cc]    <Library>
            <Function>
            [<parameter 1>]
            [<parameter 2>]
            ....
            [<parameter n>]
```

Example:

```
CSF [cc]    S.IP.Library
            S.IP.InitODM
            F 1000
            R 1000
            1000
```

The PCD2.M480 can be equipped with two PCD7.F65x. Depending on the slot (B1 or B2), the description of the library for the initialisation of Open Data Mode must be adapted.

```
Example for slot B1: CSF [cc]    S.IP.Library_SlotB1
                                S.IP.InitODM
                                F 1000
                                R 1000
                                1000
```

```
Example for slot B2: CSF [cc]    S.IP.Library_SlotB2
                                S.IP.InitODM
                                F 2000
                                R 2000
                                1000
```

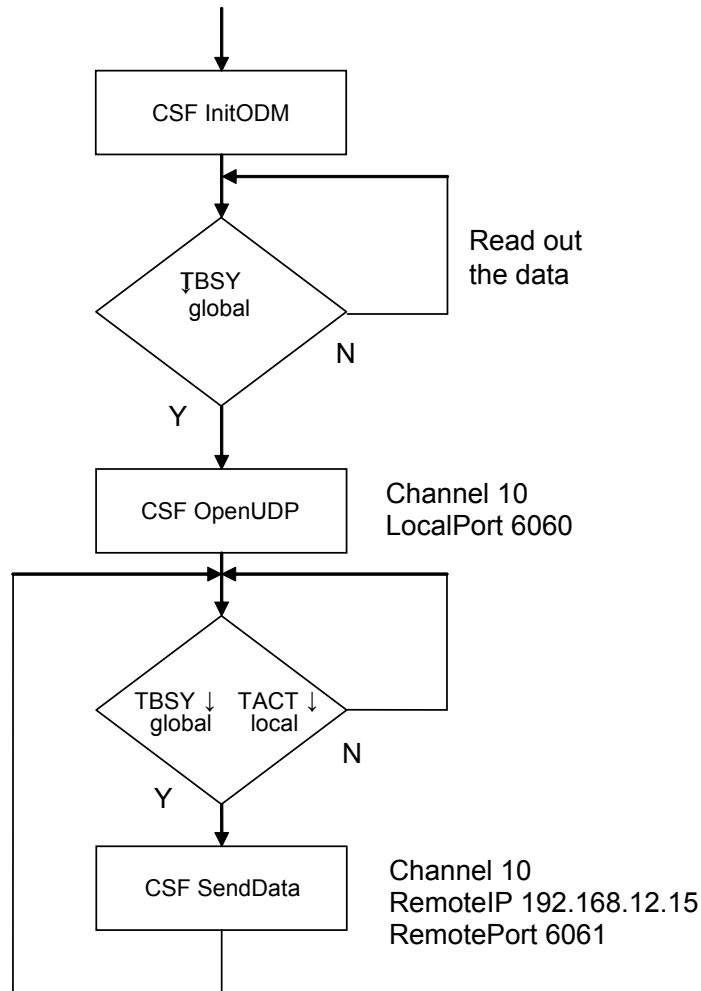
Overview of the available CSFs for Open Data Mode. Details on the different functions are provided in the following pages.

CSF 0	IPODMInit
CSF 1	IPODMOpenUDP
CSF 2	IPODMOpenClientTCP
CSF 3	IPODMOpenServerTCP
CSF 4	IPODMClose
CSF 5	IPODMConnectTCP
CSF 6	IPODMDisconnectTCP
CSF 7	IPODMGetConnectionTCP
CSF 8	IPODMAcceptTCP
CSF 9	IPODMSend
CSF 10	IPODMRead
CSF 11	IPODMSendRev
CSF 12	IPODMReadRev

Here are two examples in block diagram form to explain how these CSFs are used. To simplify the diagrams, “JR -1” steps are used for YES/NO queries.

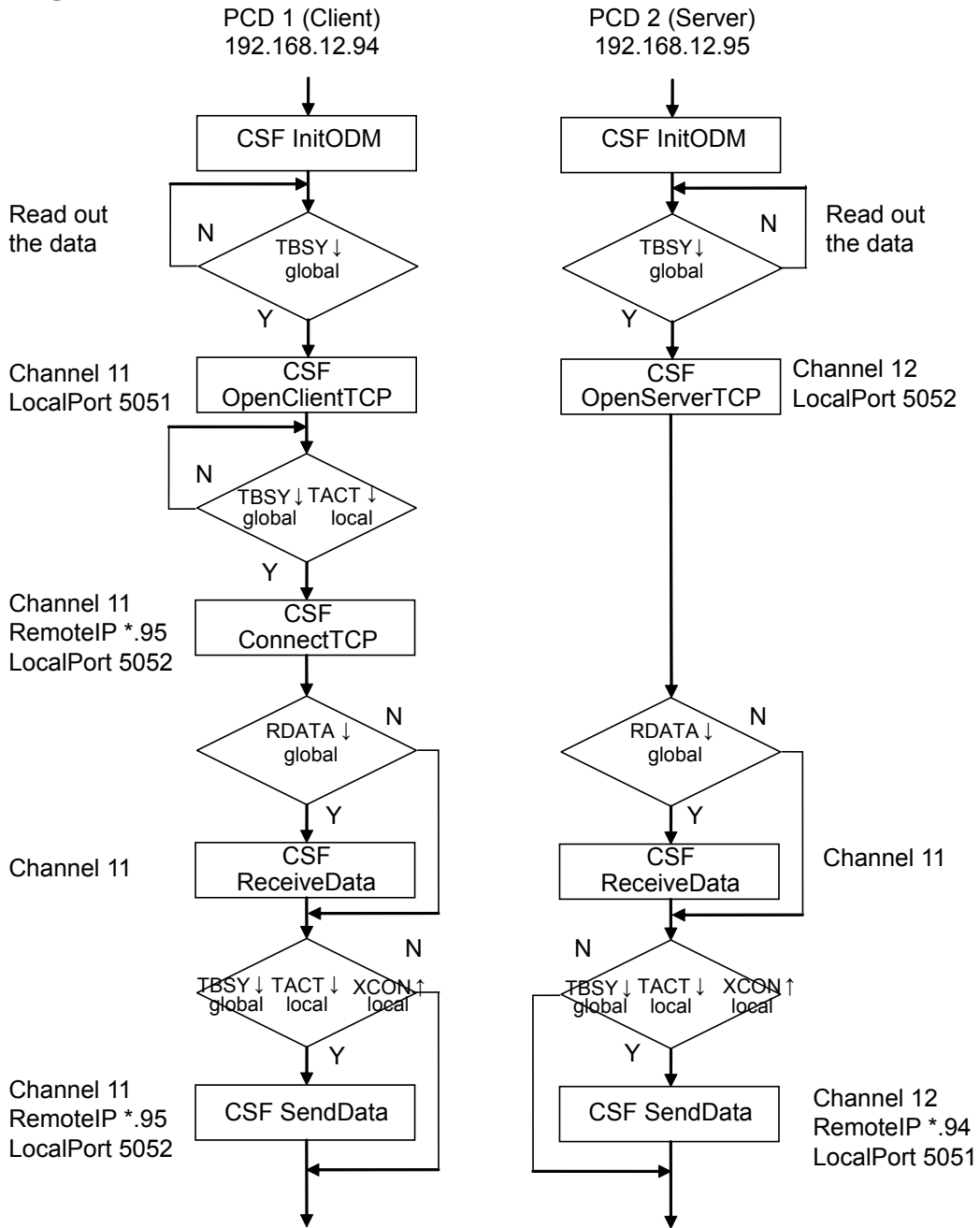
First of all, UDP communication: The “Open Data Mode” CSF is required in order to initialise Open Data Mode. As UDP is connectionless, no connection is established between the communication subscribers. There is therefore no client/server relationship and each subscriber is able to communicate with any other UDP subscriber. The only prerequisite is that the target subscriber has a UDP socket open on a port known to the sending subscriber. In the following example, it is assumed that another addressable subscriber with IP address 192.168.12.15 and an open UDP socket at port 6061 is connected to the network. As explained later in detail, handling of the diagnostic flag “RDATA” (data received) must be assigned the highest priority.

# UDP



An example of TCP communication: The “Open Data Mode” CSF is required in order to initialise Open Data Mode. As TCP is connection-oriented, the client must establish a connection to its server over which telegrams can then be exchanged. The client opens a TCP client socket using “OpenClientTCP” and the server opens a TCP server socket via “OpenServerTCP”. The server listens on this port for a connection request. This is why it is often referred to as the “Listener Socket”. The “ConnectTCP” CSF causes the client to connect to the server subscriber using 192.168.12.95 and port 5052. Once the connection is successfully established, there is essentially no longer a difference between client and server. Both subscribers can then take the initiative and send each other telegrams and requests. Both also have the ability to terminate the connection at any point. As explained later in detail, handling of the diagnostic flag “RDATA” (data received) must be assigned the highest priority.

# TCP



### 4.3.5 InitODM

Under PG5 1.4 a first provisionnal mechanism has been introduced to share the ODM mode between SBC and Engiby's libraries. Because only few user applications have been concerned by this case under PG5 1.4 this old principle is not described here.

For PG5 2.0, a more universal principle has been implemented and should now be respected by all devopers of communication application under ODM mode. This version is described in this document and allows the usage of the ODM by several libraries in parallel.

#### 4.3.5.1 Initialization of ODM mode

The ODM mode must be initialized once in the CPU for all applications by mean of a SCF instruction. This instruction cannot be placed in the user application nor in a communication library because only one will succeed in case of shared ODM.

To solve this point, the ODM initialisation is now executed by the SPM when the corresponding option is set in the Device Configurator under the Ethernet options, in the TCP/IP group

Initialize Open Data Mode	Yes
Telegram Reading Timeout	1000

By setting the the option to Yes, 3 actions are executed by SPM:

- The necessary code to initialize the ODM is generated
- The Reading timeout is defined for all applications
- A set of system symbols are published

#### 4.3.5.2 Telegram Reading timeout

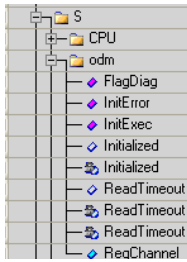
The default (and recommended value) for the Reading Timeout is 1000 milli-seconds. This value is valid for all application part using the ODM. It is the maximum time allowed for reading of a received datagram. Each application is responsible for the reading of its addressed packages. As long as a packet is not read from the reception buffer, no other ones (for other applications) can be read. Therefore, the reading timeout is a security mechanism to ensure that one application cannot lock the reception buffer used by other application.

However, if an application is not anymore reading received datagram it will drastically slow down the communication of other application parts and may even cause to lost of packages addressed to others.

Therefore, an application using the shared ODM must allways check for incoming packages and read them as fast as possible. If the immediate processing of the package is not possible the package should be read and stored in a temporary buffer (or ignored with error indication).

### 4.3.5.3 System symbols

This is the list of published system symbols.



4

#### Description

s.odm.FlagDiag	Base address of 8 diagnostic flags. See Ethernet manual for details.
s.odm.FlagDiag+0	Transmission busy
s.odm.FlagDiag+1	Reception busy
s.odm.FlagDiag+2	Connection event received
s.odm.InitError	Initialization failed. Valid only if InitExec is set
s.odm.InitExec	Initialization has been executed
s.odm.Initialized	ODM initialization option is set
s.odm.ReadTimeout	Reading timeout value in milli-sec
s.odm.RegChannel	Register indicating the concerned channel by a received package

The application using the shared ODM should first check that the ODM option has been set and issue a message (\$warning, \$error or even \$fatal) if it's not the case. Assembler directives should also be used to avoid further assembler errors if ODM is not initialized. This makes the understanding and correction of the problem easier for users.

#### Example :

```

$ifndef s.odm.initialized
    $error Driver xyz, ODM not initialized. See Device configurator.
    exitm ;exit macro to avoid further errors
$else
$if s.odm.initialized = 0
    $error Driver xyz, ODM not initialized. See Device configurator.
    exitm ;exit macro to avoid further errors
$endif
$endif

```

#### 4.3.5.4 Initialisation execution and error

Before to use the shared ODM, the application must ensure that the initialisation is executed without error. Because we cannot predict where the code in \$init section is placed, this check must be done in COB segment using the both system symbols: s.odm.InitExec and s.odm.InitError.

Example:

```
$cobseg
    sth    s.odm.InitExec    ;;Executed
    anl    s.odm.InitError   ;;wihtout error
    jr     l l_skip         ;;->Not yet initialized or error
                                ;;Else ready to use ODM
$endcobseg
```

4

#### 4.3.5.5 Choose of ODM channel

Don't mix up S-Bus channels indication (8 or 9) in the Device configurator. Those channels are only valid for S-Bus and select the accessed Ethernet card.

For the ODM mode, each CPU has a limited number of logical channels. Each driver must use a different channel. Therefore, the ODM channel number must be adjustable by the user to be able to avoid channel collisions.

The number of channels are specificated in chapter 4.3.7

**Mechanism for a dynamic allocation of ODM channels is under study. It is not yet introduced in this document.**

In the meantime, the following rules apply:

- Channels 1 to 10 (adjustable) should be used for user applications and third party libraries.
- Channels 11 to 19 reserved for future use
- Channels 20 to 29 (not adjustable) are used for SBC standard Saia PG5<sup>®</sup> FBoxes (WAA library)

#### 4.3.5.6 Package reception

As explained above, each application using the shared ODM must cyclically check the reception of packages and read the received packages and only packages addressed to its channel as fast as possible.

The application must check the channel which a package is address to using the system symbol `s.odm.RegChannel`.

Example:

```
sth    s.odm.FlagDiag+1
jr     l l_no_data ;;->No reception

cmp    .s.odm.RegChannel
       My_Channel
acc    z
jr     l l_no_data ;;->Not for my channel

CSF    ReceiveData
```

4

#### 4.3.5.7 Connection events reception

The same rules apply for the reception of connection events if you are using the TCP client connection filter.

The application must cyclically check the reception of events and read the received events and only events addressed to its channel as fast as possible.

The application must check the channel which an event is address to using the system symbol `s.odm.RegChannel`.

Note that the same register is used to indicate the channel for data package and connection evnets. The FW ensures that only one of both flags (data reception and event reception) is set at the same time. Therefore, it is also important to allways read connection events even if the events can be ignored in some situations.

Example:

```
sth    s.odm.FlagDiag+2
jr     l l_no_event ;;->No event reception

cmp    s.odm.RegChannel
       My_Channel
acc    z
jr     l l_no_event ;;->Not for my channel

CSF    Get connection event
```



### 4.3.6 Diagnostics

#### Global diagnostic flags

Address	Name	Description:
Xxxx	TBSY	Transmitter busy
xxxx + 1	RDATA	Receive data server
xxxx + 2	RCON	Connection
xxxx + 3	Reserved	
xxxx + 4	Reserved	
xxxx + 5	Reserved	
xxxx + 6	XBSY	Physical link
xxxx + 7	Reserved	

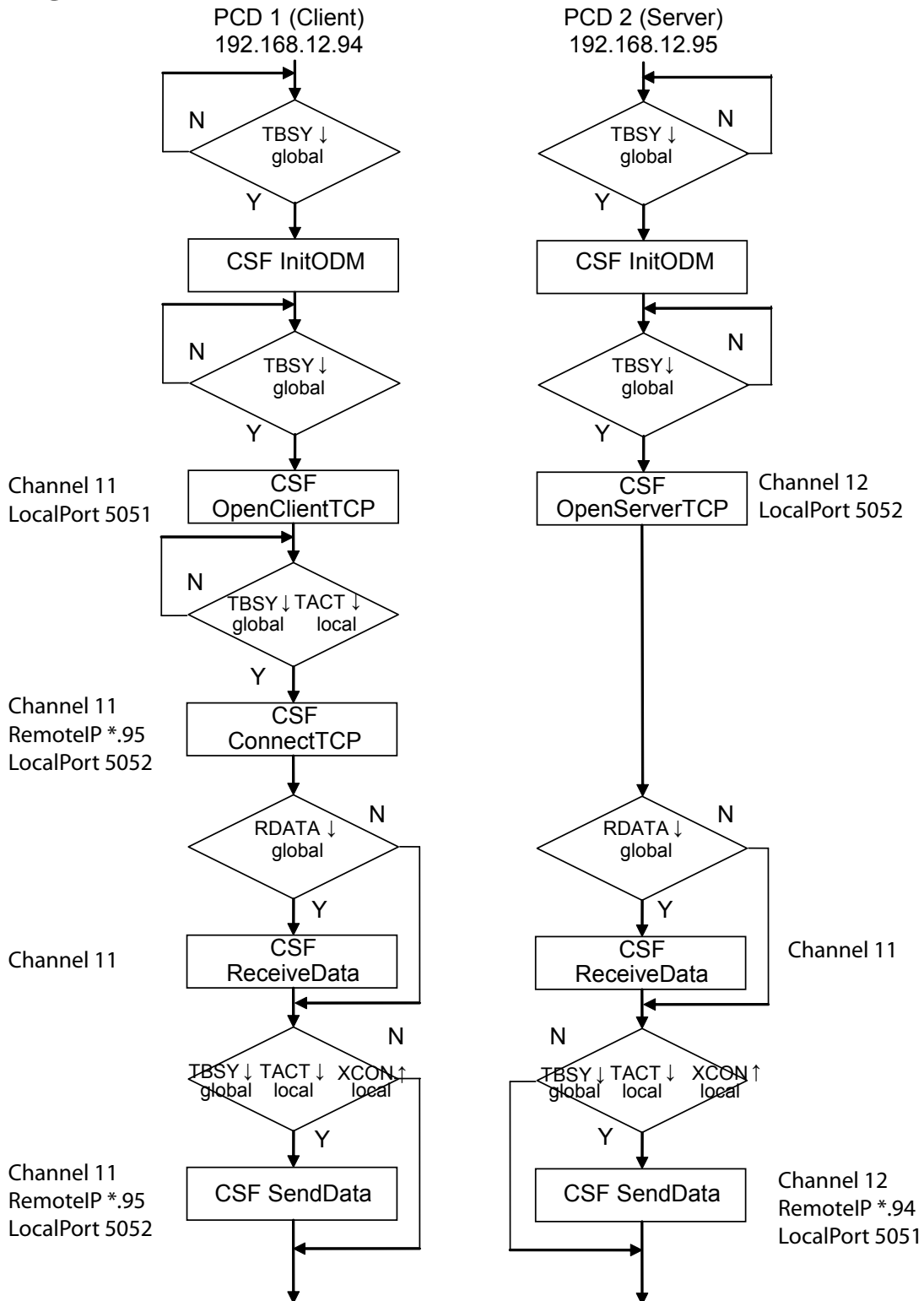
**Transmitter busy (TBSY)**↑ Set if there is a telegram in the send buffer. It is not possible to send data to any other channel when this flag is set.

**Receive data (RDATA)**↑ Set if there is a telegram in the receive buffer. This is the most important flag for Open Data Mode and must accordingly be handled with the highest priority. As soon as there is data in the receive mailbox on the PCD - as indicated by RDATA "H" (high) - the mailbox must be read by the PCD, the data processed, and the mailbox vacated again for all channels by calling the Receive command. The send and receive buffers can hold a maximum of 1536 bytes (720 bytes with PCD7.F650/F655) of user data per transaction (send/receive).  
When a data telegram is received - RDATA "H" (high) - the contents of the channel register (global diagnostic on InitODM) shows which channel the data has been sent on. Important in this respect: If the local station is communicating with more than one opposite station (for example, a TCP server which is connected to several TCP clients), before the "Receive Data / Receive Data Rev" CSF is called, the contents of the channel register must be cached. This means that after the data has been retrieved, the copy of the channel register can be used again, because, right after the "Receive Data / Receive Data Rev" CSF is called, the next incoming telegram moves up, and the contents of the channel register can be modified.



For information on this, see also the entry FAQ #100584 "Receive data in Open Data Mode".

# TCP



**Receiver connection (RCON)↑**

Set if a change occurs to a connection with a TCP server and the "GetConnectionTCP" function must be executed at the same time.

**Physical Link (XBSY)** Indicates whether a physical link to the next hub/switch exists or not. Set to “H” (high) when the Ethernet TCP/IP module detects a physical link and to “L”(low) when the physical link is lost.

**Channel diagnostic flags**

In Open Data Mode, the user programs almost directly into an abstraction layer via the Berkeley sockets. In order to simplify and abstract programming, so-called channels are used. Practically every socket on the Ethernet TCP/IP module (UDP and TCP Open Data Mode) has a virtual channel which is used for communication. Each channel has its own diagnostic flags and diagnostic register. The channels are accessed using the functions “OpenUDP”, “OpenClientTCP”, “OpenServerTCP”, “Close”, “ConnectTCP”, “DisconnectTCP”, “AcceptConnectionTCP”, “SendData”, “SendDataRev”, “ReceiveData” and “ReceiveDataRev”.

Address	Name	Description:
xxxx	RDIA	Receiver diagnostic
xxxx + 1	TACT	Transmitter active
xxxx + 2	TDIA	Transmitter diagnostic
xxxx + 3	XCON	Port connected
xxxx + 4	NEXE	Not executed
xxxx + 5	Reserved	
xxxx + 6	Reserved	
xxxx + 7	Reserved	

**Receiver diagnostic (RDIA)↑** The diagnostic register contains a receiver diagnostic. The flag is set to “H” if an error is detected on receiving a telegram. You will find an in-depth description of the error in the diagnostic register (bits 0 to 15). The flag is reset as soon as all receiver diagnostic bits in the diagnostic register have been reset.

**Transmitter active (TACT)↑** Set if a telegram is in the send buffer. It is not possible to send other data on this channel or to connect or disconnect this channel until TACT is no longer in effect.

**Transmitter diagnostic (TDIA)↑** The diagnostic register contains a transmitter diagnostic. It is set to “H” if an error is detected while sending a telegram. An in-depth description of the error can be found in the diagnostic register (bits 16 to 31). The flag is reset as soon as all transmitter diagnostic bits in the diagnostic register have been reset.

**Port connected (XCON)↑** The TCP port is connected to another TCP station.

**Not executed (NEXE)↑** Set if an instruction is incomplete or was not executed. The flag is reset by the next successful communication instruction.

**Channel diagnostic register**

	Bit	Designation	Description
RECEIVER	0	Read timeout error	A telegram was not read from the buffer.
	1	Not used	
	2	Not used	
	3	Not used	
	4	Not used	
	5	Not used	
	6	Not used	
	7	Not used	
	8	Length error	The telegram received is longer than the buffer
	9	Not used	
	10	Address error	Error while converting the IP address from the buffer
	11	Not used	
	12	Range error	Invalid element address
	13	Status error	Invalid status
	14	Rx Mailbox error	Mailbox contains no data
15	Rx Channel error	Incorrect channel number	
TRANSMITTER	16	Not used	
	17	Not used	
	18	Not used	
	19	Not used	
	20	Not used	
	21	Not used	
	22	Not used	
	23	Stat. not present	Destination station does not exist
	24	Not used	
	25	Not used	
	26	Not used	
	27	Not used	
	28	Range error	Invalid element address
	29	Add error	Port does not exist or port and address do not match
	30	No connection	The connection to this station is not open
	31	Program error	Impermissible send attempt

Each bit which is set to "H" (high) in the diagnostic register remains set until it is reset manually by the user program or debugger manuell zurückgesetzt wird.

**Read timeout error (bit 0)** Set to "H" if a telegram is not read from the buffer. Buffer timeout is too short or channel was not read.

**Length Error (bit 8)** Set to "H" if the telegram exceeds the length of the buffer.

- Address error (bit 10)**            Set to “H” if an error occurs during the conversion of the IP address from the buffer:
  - Conversion in register
  - Text too short
- Range error (bit 12)**            Set to “H” if the element address is invalid.
- Mailbox error (bit 14)**          Set to “H” if the mailbox contains no data.
- Channel error (bit 15)**          Set to “H” if the channel number is incorrect.
- Target not present (bit 23)**    Set to “H” if the destination station cannot be contacted on the network. For example, if there is no physical link to the next hub/switch and a telegram is sent by UDP (not PCD7. F655), or if the transmitting socket in the TCP/IP stack reports an error, causing the telegram not to be sent on the Ethernet.
- Range error (bit 28)**            Set to “H” if the element address is invalid.
- TxNode Error (bit 29)**          Set to “H” if the node does not exist.
- TxNode Error (bit 29)**          Set to “H” if the node does not exist.
- TxNode Error (bit 29)**          Set to “H” if the node does not exist.
- No connection (bit 30)**        Set to “H” if the connection to this station is not open.

**General determination of the channel management for OpenUDP, OpenClientTCP and OpenServerTCP**

**4.3.7 Amount of ODM channels**

The total number of channels permitted per PCD (TCP and UDP) depends on the PCD system. They are the following:

PCD system	PCD1.M130	PCD2.M150	PCDx.M170	PCD2.M480
number of channels	16	32	32	32
PCD system	PCD3.Mxxxx	PCD1.M2xx0	PCD2.M5	
number of channels	32	32	32	

It should also be mentioned that S-Bus UDP requires a single channel if both the input port (S-Bus server) and output port (S-Bus client) are set to 5050. If the S-Bus server and S-Bus client are using different ports, the system needs two channels. The additional channels can then be used in Open Data Mode, meaning that, for this table, all channels opened using “OpenUDP”, “OpenTCPClient” and “OpenTCPServer” are considered jointly.

### 4.3.8 OpenUDP

Opens a UDP channel in IP Open Data Mode (corresponds with a “socket” and “bind” in the Berkeley standard). After this call, telegrams can be sent via this UDP channel.

This function can be executed only if TBSY is set to “L” (low). CSF

```
[cc] S.IP.Library      ; IP library
      S.IP.OpenUDP    ; “Open UDP channel” function
      → LocalPort      ; the local IP port over which data is
                        ; sent/received (R/K)
      → Ch_Diag_Flag   ; basis of the channel diagnostic flags
                        ; (8 flags) (F)
      → Ch_Diag_Register ; channel diagnostic register (R)
```

4

**LocalPort:** selection of the local port for Open Data Mode UDP communication  
 0 = a free port is selected on the system  
 X = direct assignment of the local port by the user  
 Local port 5050 is reserved for S-Bus UDP mode.

**TACT↑** when the command “OpenUDP” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TACT↓** after an ACK response has been received from the Ethernet TCP/IP module. A new Open Data Mode command can be executed.

**NEXE↑** If, for example the local IP port is already in use.

**Example:** CSF [cc] S.IP.Library ; library  
 S.IP.OpenUDP ; function  
 10 ; channel 10  
 5000 ; Port 5000  
 F 1010 ; basis of the channel diag.  
 ; nose-flags  
 R 1 ; channel diagnostic register

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

### 4.3.9 OpenClientTCP

Opens a TCP channel in IP Open Data Mode (corresponds with a “socket” and “bind” in the Berkeley standard). After this call, the TCP client can connect to a TCP server.



This function can be executed only if TBSY is set to “L” (low).

CSF	[cc]	S.IP.Library	; IP library
		S.IP.OpenClientTCP	; “Open TCP channel” function
→		LocalPort	; the local IP port over which data is sent/received (R/K)
→		Ch_Diag_Flag	; basis of the channel diagnostic flags; (8 flags) (F)
→		Ch_Diag_Register	; channel diagnostic register (R) Conn_Tout
→		Conn_Tout	; connection timeout: 0=none, x=sec. (R/K)

4

**LocalPort:** selection of the local port for Open Data Mode TCP communication  
 0 = a free port is selected on the system  
 X = direct assignment of the local port by the user  
 Although local port 5050 is reserved for S-Bus UDP communication, this port can still be used in TCP.

**Conn\_Tout:** if Conn\_Tout (seconds) is exceeded without a telegram being received via the channel, the channel is closed (unless something is being received).

- 0 = no timeout check
- x = timeout check every x seconds

**TACT↑** when the command “OpenClientTCP” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TACT↓** after an ACK response has been received from the Ethernet TCP/IP module. A new Open Data Mode command can be executed.

**NEXE↑** if, for example, the local IP port is already in use.

**Example:**

CSF	[cc]	S.IP.Library	; library
		S.IP.OpenClientTCP	; function
		10	; channel 10
		5555	; Port 5555
		F 1010	; basis of the channel diag. nose flags
		R 1	; channel diagnostic register
		R 2	; connection timeout

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

### 4.3.10 OpenServerTCP

Opens a TCP server channel in IP Open Data Mode (corresponds to a “socket”, “bind” and “accept” in the Berkeley standard). After this call, the TCP server is ready to accept a connection from a TCP client.



This function can be executed only if TBSY is set to “L” (low).

```
CSF [cc]  S.IP.Library           ; IP library
          S.IP.OpenServerTCP ; “Open UDP channel” function
          → Channel             ; channel no. (R/K)
          → LocalPort          ; the local IP port over which data is
                               ; sent/received (R/K)
          → Ch_Diag_Flag       ; basis of the channel diagnostic flags
                               ; (8 flags) (F)
          → Ch_Diag_Register   ; channel diagnostic register (R) Conn_Tout
          → Connection_filter  ; connection filter (R/K)
          → Conn_Tout          ; connection timeout: 0=none, x=sec.
                               ; (R/K)
```

4

**LocalPort:** selection of the local port for Open Data Mode UDP communication  
 0 = a free port is selected on the system  
 X = direct assignment of the local port by the user  
 Although local port 5050 is reserved for S-Bus UDP communication, this port can still be used in TCP.

**Connection\_filter** ● 0 = no filter, no connection info available (automatically accept all clients)  
 (only on TCP server) ● 1 = no filter, connection info available (automatically accept all clients)  
 ● 2 = Filter request to accept the client, connection info available

**Conn\_Tout:** if Conn\_Tout (seconds) is exceeded without a telegram being received via the channel, the channel is closed (unless something is being received).

- 0 = no timeout check
- x = timeout check every x seconds

**TACT↑** when the command “OpenUDP” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TACT↓** after an ACK response has been received from the Ethernet TCP/IP module. A new Open Data Mode command can be executed.

**NEXE↑** if, for example, the local IP port is already in use.

```
Example: CSF [cc]  S.IP.Library           ; library
          S.IP.OpenServerTCP ; function
          10             ; channel 10
          5432           ; Port 5432
          F 1010        ; basis of the channel diag.
                               ; nose flags
          R 1           ; channel diagnostic register

          0             ; no filter
          R 2           ; connection timeout
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.



**4.3.11 Close**

Closes a channel in IP Open Data Mode (corresponds to a “closesocket” in the Berkeley standard).



This function can be executed only if TACT is set to “L” (low).

```
CSF [cc] S.IP.Library ; IP library
          S.IP.Close ; “Close channel” function
          → Channel ; channel no. (R/K)
```

```
Example: CSF [cc] S.IP.Library ; library
            S.IP.Close ; function
            10 ; channel 10
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

The diagnostic flags and diagnostic register which were temporarily defined for this channel should not be used anymore after the “Close” until they have been reinitialised and reassigned. The “Close” CSF terminates the connection between the two stations on PCD7.F650/1/2 immediately. In this case, the PCD7.F655 behaves a little differently. The difference is described in the following example:

A TCP client connects to a TCP server and sends a telegram. The global diagnostic flag RDATA↑ is set on the server. However, the server does not read the telegram at that precise moment. If the client then issues a “Close”, the socket on the client is closed immediately. On the server, the socket is closed only when the above mentioned telegram has been read and the global diagnostic flag RDATA↑ has been set to “low”. This prevents any telegrams from being lost in the TCP server when the connection is closed.



**4.3.12 ConnectTCP**

Connects a client TCP channel in IP Open Data Mode to a server TCP channel (corresponds with a “connect” in the Berkeley standard). After this call, communication can occur over the TCP connection.



This function can be executed only if TACT and TBSY are set to “L” (low).

```
CSF [cc]  S.IP.Library           ; IP library
           S.IP.ConnectTCP       ; “Connect TCP channel” function
           → Channel             ; channel no. (R/K)
           → RemoteIP/Node       ; IP address of the remote server
                                   ; (R/K/X)*
           → Remote Port         ; IP port of the remote server (R/K)
           *) see chapter: IP address decoding
```

4

- TBSY**↑     when the command “ConnectTCP” has been stored in the communication mailbox of the Ethernet TCP/IP module.
- TACT**↑     when the command “ConnectTCP” has been stored in the communication mailbox of the Ethernet TCP/IP module.
- TBSY**↓     after the Ethernet TCP/IP module has read the “ConnectTCP“ from the communication mailbox. A new Open Data Mode command can be executed.
- TACT**↓     after an ACK response has been received from the Ethernet TCP/IP module. A new Open Data Mode command can be executed.
- XCON**↑     when the “Connected Event” is returned by the Ethernet TCP/IP module. There is now a connection between the TCP client and the TCP server.

**Example:**

```
CSF [cc]  S.IP.Library           ; library
           S.IP.ConnectTCP       ; function
           10                     ; channel 10
           R5                      ; remote IP address
           5555                    ; Port 5555
```

**Flags:**     If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

Until the ConnectTCP CSF has been completed, its parameters “RemoteIP/Node” and “RemotePort” must not be changed.

### 4.3.13 DisconnectTCP

Disconnects a TCP channel in IP Open Data Mode. (corresponds with a “closesocket” in the Berkeley standard). This function can be executed on a TCP client and TCP server channel.



This function can be executed only if TACT and TBSY are set to “L” (low)

```
CSF [cc] S.IP.Library ; IP library
          S.IP.DisconnectTCP ; “Disconnect TCP channel” function
          → Channel ; channel no. (R/K)
          → RemoteIP/Node ; IP address of the remote server
                               ; (R/K/X)*
          → Remote Port ; IP port of the remote server (R/K)
          *) see chapter: IP address decoding
```

**TBSY**↑ when the command “DisconnectTCP” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TACT**↑ when the command “DisconnectTCP” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TBSY**↓ after the Ethernet TCP/IP module has read the “ConnectTCP” from the communication mailbox. A new Open Data Mode command can be executed.

**TACT**↓ after an ACK response has been received from the Ethernet TCP/IP module. A new Open Data Mode command can be executed.

**XCON**↑ when the “Connected Event” is returned by the Ethernet TCP/IP module. There is now a connection between the TCP client and the TCP server.

**Example:** CSF [cc] S.IP.Library ; library  
 S.IP.DisconnectTCP ; function  
 10 ; channel 10  
 R5 ; remote IP address  
 5555 ; Port 5555

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

The “Disconnect” CSF immediately terminates the connection between the two stations on PCD7.F650/1/2 and the XCON ↓ diagnostic flag instantly displays the connection status. In this case, the PCD7.F655 behaves a little differently. The difference is described in the following example: A TCP client connects to a TCP server and sends a telegram. The global diagnostic flag RDATA ↑ is set on the server. However, the server does not read the telegram at that precise moment. If the client then issues a “Disconnect”, the XCON ↓ diagnostic flag on the client is immediately set to “low”. On the server, XCON ↓ is only set to “low” when the above mentioned telegram has been read and the global diagnostic flag RDATA ↑ has been set to “low”. This prevents any telegrams from being lost in the TCP server when the connection is terminated.

#### 4.3.14 GetConnectionTCP

Reads the connection data for a TCP channel in IP Open Data Mode

**RCON**↑ displays changes to a TCP client or server. The change can be read from the server using the “GetConnectionTCP” command. The information about the status “Connection requested/ Connected/ Disconnected” can be read only if the filter in the previously defined TCP server is set to 1 or 2. “GetConnectionTCP” can be executed on only one TCP server channel. If several channels are being used, the channel which is specified in the channel register should be accessed.

```
CSF [cc] S.IP.Library           ; IP library
          S.IP.GetconnectionTCP ; “TCP-channel connection data” function
          → Channel             ; channel no. (R/K)
          → RemoteIP/Node       ; IP address of the remote server
                                     ; (R/K/X)*
          → Remote Port         ; IP port of the remote server (R/K)
          Status                ; connection status**
```

\*) see chapter: IP address decoding

\*\*) Status: 0=client requesting connection and awaiting acceptance  
from the server  
1=connected  
2=disconnected

**RCON**↓ after the GetconnectionTCP command is executed

```
Example: CSF [cc] S.IP.Library           ; library
          S.IP.GetconnectionTCP         ; function
          10                            ; channel 10
          R100                           ; remote IP address
          R101                           ; remote port
          R102                           ; status
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

### 4.3.15 AcceptConnectionTCP

Accepts a connection on a server TCP channel in IP Open Data Mode. This function can be used only in conjunction with a TCP server station which has been defined in "OpenServerTCP" with connection filter options "1" or "2".



This function can be executed only if TACT and TBSY are set to "L" (low)

```
CSF [cc]  S.IP.Library           ; IP library
           S.IP.AcceptconnectionTCP ; "TCP-channel connection data" function
           → Channel              ; channel no. (R/K)
           → Status                ; connection status*
           *) Status:             0=accept connection
                                   1=reject connection
```

4

**Example:**

```
CSF [cc]  S.IP.Library           ; library
           S.IP.AcceptconnectionTCP ; function
           10                      ; channel 10
           0                        ; status=accept connection
           ; nehmen
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

**4.3.16 SendData**

Sends data over a channel in IP Open Data Mode (corresponds with a “send” for TCP or a “sendto” for UDP in the Berkeley standard).



This function can be executed only if TBSY and TACT are set to “L” (low).

```
CSF [cc] S.IP.Library           ; IP library
          S.IP.SendData        ; “Send data” function
          → Channel            ; channel no. (R/K)
          → RemoteIP/Node      ; IP address for sending data
                                   ; (R/K/X)*
          → Remote Port        ; IP port of the remote server (R/K)
          → Datalength         ; length of data to be sent (R/K)**
          → Data                ; data buffer (R/X/DB)
```

4

\*) see chapter: IP address decoding

\*\* ) The max. length for sent data is 1536 bytes

**TACT**↑ when the command “SendData” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TBSY**↑ when the command “SendData” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TACT**↓ after an ACK response has been received from the Ethernet TCP/IP module.

**TBSY**↓ after the Ethernet TCP/IP module has read the “SendData“ from the communication mailbox. A new Open Data Mode command can be executed.**Example:**

```
CSF [cc] S.IP.Library           ; library
          S.IP.SendData; function
          10                    ; channel 10
          R100                   ; remote IP addre
          R101                    ; remote port
          100                     ; length of data for transmission
          R1000                   ; start of the send buffer
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

The data buffer must not be modified before the “SendData” command has been fully completed.

**4.3.17 SendDataRev**

Sends data in reverse byte sequence over a channel in IP Open Data Mode (corresponds with a “send” for TCP or a “sendto” for UDP in the Berkeley standard).

SendDataRev swaps the bytes around if the remote station is using the “Intel” format (see following table).



This function can be executed only if TBSY and TACT are set to “L” (low).

```
CSF [cc] S.IP.Library ; IP library
          S.IP.SendDataRev ; “Send data” function
          → Channel ; channel no. (R/K)
          → RemoteIP/Node ; IP address for sending data
          ; (R/K/X)*
          → Remote Port ; IP port of the remote server (R/K)
          → Datalength ; length of data to be sent (R/K)**
          → Data ; data buffer (R/X/DB)
```



\*) see chapter: IP address decoding  
 \*\*) The max. length for sent data is 1536 bytes

**TACT**↑ when the command “SendData” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TBSY**↑ when the command “SendData” has been stored in the communication mailbox of the Ethernet TCP/IP module.

**TACT**↓ after an ACK response has been received from the Ethernet TCP/IP module.

**TBSY**↓ after the Ethernet TCP/IP module has read the “SendData” from the communication mailbox. A new Open Data Mode command can be executed.

**Example:**

```
CSF [cc] S.IP.Library ; library
          S.IP.SendDataRev ; function
          10 ; channel 10
          R100R ; remote IP address
          R101 ; remote port
          100 ; length of data for transmission
          R1000 ; start of the send buffer
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

The data buffer must not be modified before the “SendData” command has been fully completed.

### 4.3.18 ReceiveData

Receives data over a channel in IP Open Data Mode

This function can be executed only if RDATA is set to “H” (high). If several channels are in use, the channel on which the data arrived must be specified in the channel parameter (channel register). Read the contents of the channel register (configured in the InitODM) immediately before calling the “ReceiveData” CSF and copy it to use as the “Channel” in the command.

```
CSF [cc] S.IP.Library           ; IP library
          S.IP.ReceiveData      ; function
          → Channel              ; channel no. (R/K)
          ← RemoteIP/Node       ; IP address for sending data
                                   ; (R/K/X)*
          ← Remote Port         ; IP port of the remote server (R/K)
          → Max_Datalength      ; max. length of the data buffer in bytes
                                   ; (0=no check)**
          → Datalength          ; length of data to be sent (R/K)**
          → Data                 ; data buffer (R/X/DB)
          *) see chapter: IP address decoding
          **) The max. length for sent data is 1536 bytes
```

**RDATA**↓ after the ReceiveData command is executed

```
Example: CSF [cc] S.IP.Library           ; library
              S.IP.ReceiveData      ; function
              10                     ; channel 10
              R100R                   ; remote IP address
              R101                     ; remote port
              100                      ; data length
              R1                       ; data received in bytes
              R1000                    ; start of the send buffer
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.



**4.3.19 ReceiveDataRev**

Receives data in reverse byte sequence over an IP channel in Open Data Mode. ReceiveDataRev swaps the bytes around if the remote station is using the “Intel” format (see following table).

This function can be executed only if RDATA is set to “H” (high). If several channels are in use, the channel on which the data arrived must be specified in the channel parameter (channel register). Read the contents of the channel register (configured in the InitODM) immediately before calling the “ReceiveData” CSF, and copy it to use as the “Channel” in the command.

```

CSF [cc]  S.IP.Library           ; IP library
          S.IP.ReceiveDataRev ; “Receive data” function
→ Channel           ; channel no. (R/K)
← RemoteIP/Node    ; IP address for sending data
                  ; (R/K/X)*
← Remote Port      ; IP port of the remote server (R/K)
→ Max_ Datalength  ; max. length of the data buffer in bytes
                  ; (0=no check)**
→ Datalength       ; length of data received (R/K)**
→ Data             ; receive data buffer (R/X/DB)
    
```

\*) see chapter: IP address decoding  
 \*\*) The max. length for sent data is 1536 bytes

**RDATA↓** after the ReceiveData command is executed

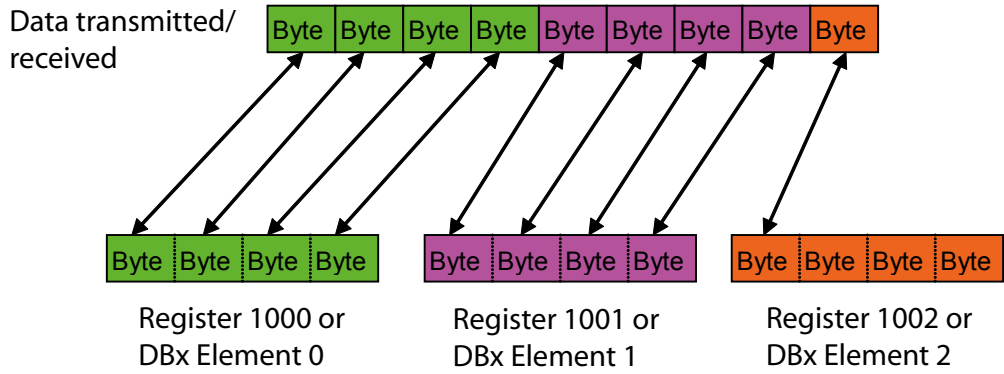
```

Example: CSF [cc]  S.IP.Library           ; library
              S.IP.ReceiveDataRev ; function
              10                   ; channel 10
              R100R                 ; remote IP address
              R101                  ; remote port
              100                   ; data length
              R1                    ; data received in bytes
                                   ; Bytes
              R1000                 ; start of the receive buffer
    
```

**Flags:** If the firmware does not offer IP Open Data Mode support, the Error (E) flag is set.

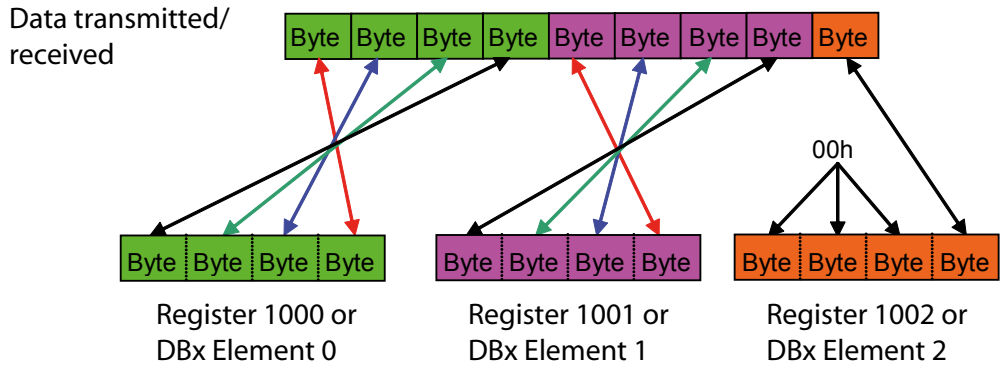
### 4.3.20 Byte swapping

Example: 9 bytes need to be sent/received.  
→ Without byte swapping (SendData / ReceiveData):



4

→ With byte swapping (SendData / ReceiveData):



If the buffer contains text, the bytes will never be swapped.

### 4.3.21 IP address decoding

The IP address value may be entered in text, in a register or as a constant. The IP address may also identify a node in a register or a constant value. A constant value can be only one node.

#### ***IP address in text:***

The IP address is represented in text in the form of 4 decimal numbers separated by dots, e.g. "192.168.12.14"

#### ***IP address in a register:***

The IP address can also be represented in a register. If the higher value of the 4 bytes is zero, a node is displayed. If the higher value of the 4 bytes is not zero, the register encodes the address with 4 hexadecimal numbers:

aa	bb	cc	dd
----	----	----	----

The IP address is shown in the receipt- and connection information of the above-mentioned format.

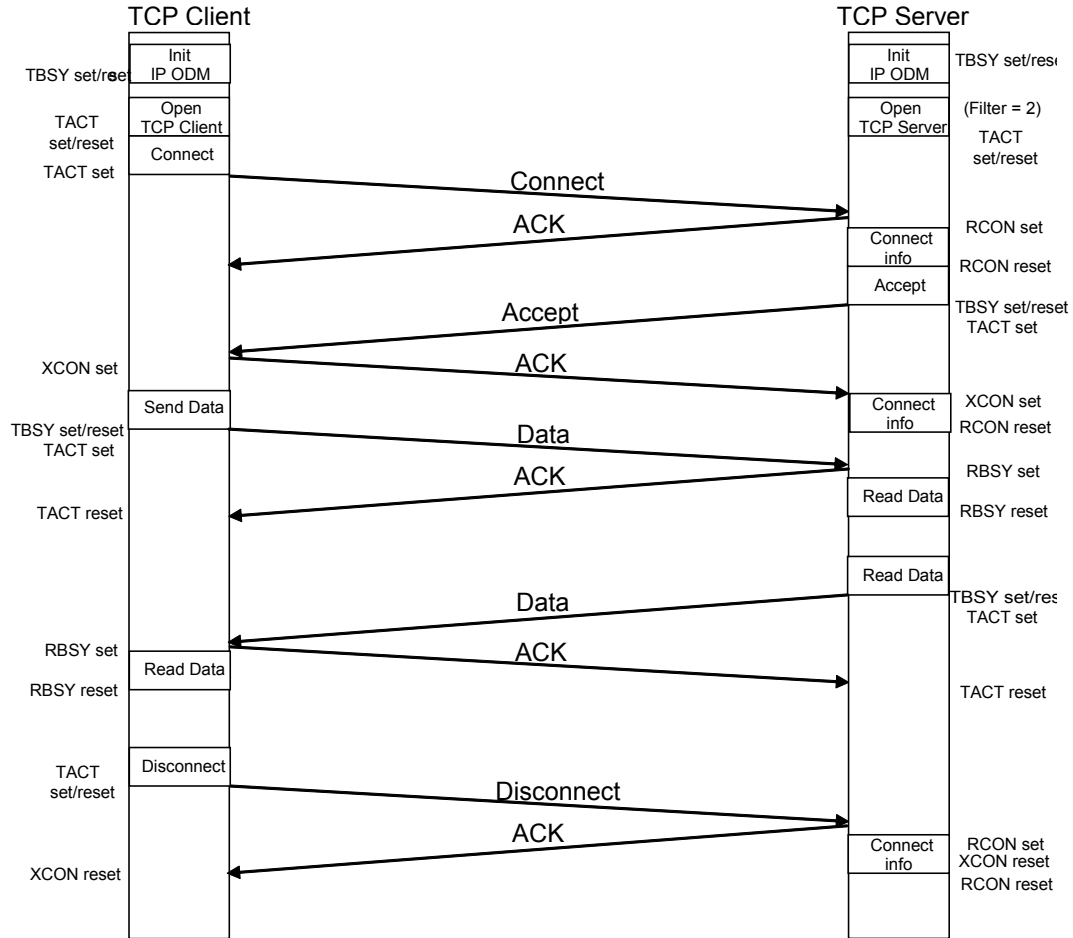
Example: 0C0A80C0Eh for the IP address 192.168.12.14

#### ***IP address in a constant:***

If a constant is used as the encoding medium, the IP address always represents the node number.

**4.3.22 A typical TCP connection process**

A TCP client (172.16.1.142) connects to a TCP server (172.16.1.141) and sends two telegrams.



Or as viewed using an Ethernet analyzer (in this case, Wireshark). In this case, only a single telegram is sent.

No. -	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.1.142	172.16.1.141	TCP	5050 > 5050 [SYN] Seq=0 Ack=0 Win=65535 Len=0 MSS=1460
2	0.001275	172.16.1.141	172.16.1.142	TCP	5050 > 5050 [SYN, ACK] Seq=0 Ack=1 Win=720 Len=0 MSS=1460
3	0.002324	172.16.1.142	172.16.1.141	TCP	5050 > 5050 [ACK] Seq=1 Ack=1 Win=720 Len=0
4	8.359572	172.16.1.142	172.16.1.141	TCP	5050 > 5050 [PSH, ACK] Seq=1 Ack=1 Win=720 Len=4
5	8.733767	172.16.1.141	172.16.1.142	TCP	5050 > 5050 [ACK] Seq=1 Ack=5 Win=720 Len=0
6	17.921981	172.16.1.142	172.16.1.141	TCP	5050 > 5050 [FIN, ACK] Seq=5 Ack=1 Win=720 Len=0
7	17.923051	172.16.1.141	172.16.1.142	TCP	5050 > 5050 [ACK] Seq=1 Ack=6 Win=720 Len=0
8	17.924448	172.16.1.141	172.16.1.142	TCP	5050 > 5050 [FIN, ACK] Seq=1 Ack=6 Win=720 Len=0
9	17.925515	172.16.1.142	172.16.1.141	TCP	5050 > 5050 [ACK] Seq=6 Ack=2 Win=720 Len=0

## 4.4 Additional CSFs

As has already been detailed in the information on Open Data Mode, “Call System Functions” provide access to special functionalities of the PCD7.F65x. Here is a list of the additional CSFs.

### 4.4.1 CSF NA Reset

This CSF triggers a hard reset of the PCD7.F65x. This the same as turning the power to the PCD7.F65x off, then on.

```
CSF      [cc]  S.IP.Library          ; IP library
          S.IP.NAReset ; "NAReset" function
          ←    Status              ; status of the PCD7.F65x (R)
```

4

The status may be one of the following values:

- 0=Reset in progress
- 1=Reset has been performed and the PCD7.F65x is ready for use again
- 2=PCD7.F65x is not present, no signature available
- 3=Status register is outside of the register limits
- 4=Mailbox communication timeout
- 5=Outdated / incompatible firmware.

### 4.4.2 CSF SetLocalIPNode

This CSF configures the PCD7.F65x for a new IP configuration

```
CSF [cc]  S.IP.Library          ; IP library
          S.IP.SetLocalIPNode ; "SetLocalIPNode" function
          →    IP_Address        ; new IP address for the PCD7.F65x (R)
          →    Subnet_Mask       ; new subnet mask for the PCD7.F65x (R)
          →    Default_Gateway   ; new default gateway for the PCD7.F65x
                                   ; (R)
```

The new IP address is updated on the PCD7.F650/1/2 without the need to power it off and on. In order to change the subnet mask and default gateway, the PCD7.F650/1/2 must be restarted.

On the PCD7.F655, it is possible to modify and update all three IP parameters during operation.

### 4.4.3 CSF IPPhyConfig

This CSF configures the Ethernet Physical Chip on the PCD7.F65x.

CSF [cc]	S.IP.Library	; IP library
	S.IP.IPPhyConfig	; "IPPhyConfig" function
→	Auto-negotiate	; whether auto-negotiate mode = TRUE/FALSE (R)
→	Speed	; communication speed, 10/100 Mb/s (R)
→	Duplex_mode	; duplex mode, full/half duplex (R)
←	Status	; status error message (R)

It is possible to change the configuration of the Physical Chip on the PCD7.F65x using a CSF. The Physical Chip works by default in auto-negotiation mode. This means it is capable of working at 10Mb/s or 100Mb/s, in half duplex or full duplex mode.

The Physical Chips can be configured either for auto-negotiate mode or for modes with a fixed communication speed and fixed duplex mode. The CSF returns a status.

Auto-negotiate:	"1"	to set the Physical Chip to auto-negotiation mode
	"0"	to work without auto-negotiation mode
Speed:	"1"	to set the Physical Chip permanently to 10Mb/s
	"2"	to set the Physical Chip permanently to 100Mb/s
	"3"	the Physical Chip checks for the right communication speed (10 or 100Mb/s) itself. Only permitted in auto-negotiation mode.
Duplex Mode	"1"	to set the Physical Chip permanently to half-duplex mode
	"2"	to set the Physical Chip permanently to full-duplex mode
	"3"	the Physical Chip checks for the right communication mode (half or full-duplex) itself. Only permitted in auto-negotiation mode.
Error Status	"0"	no error has occurred
	"1"	error has occurred <ul style="list-style-type: none"> <li>• in auto-negotiation mode: auto-negotiation was unsuccessful</li> <li>• not in auto-negotiation mode: it was not possible to switch the Physical Chip to the appropriate configuration.</li> </ul>

**Parameter “Autonegotiate” = “1”**

If the "Autonegotiate" parameter is set, the Physical Chip must be informed of which possible communication speeds and communication modes it should support. This is achieved by configuring its “Advertisement Capability Register”.

The contents of the box determines the configuration of the Advertisement Capability Register and defines which communication modes are permitted		Value of the communication speed parameter		
		1	2	3
Value of the duplex mode parameter	1	10Mb/s, half-duplex, fixed	100Mb/s, half-duplex, fixed	10Mb/s + 100Mb/s, half-duplex permitted
	2	10Mb/s, full-duplex, fixed	100Mb/s, full-duplex, fixed	10Mb/s + 100Mb/s, full-duplex permitted
	3	10Mb/s, half + full-duplex permitted	100Mb/s, half + full-duplex permitted	10Mb/s + 100Mb/s, half + full-duplex permitted

After the Physical Chip has been set to auto-negotiation mode by the CSF, the PCD7.F65x performs an auto-negotiation cycle with the station at the other end of the cable in order to properly initialise the link. If the station at the opposite end of the cable does not support auto-negotiation, communication will not run reliably and stably, and there may be problems. In this situation, the CSF therefore issues an error message containing the “status”. However, if the station at the other end of the cable does support auto-negotiation, the two devices negotiate the best communication mode for the situation. If the physical link is lost and a new link is established (if the PCD7.F65x is attached to another port on the switch, for example), a new auto-negotiation cycle begins.

**Parameter “Autonegotiate” = “0”**

If the "Autonegotiate" parameter is not set, the Physical Chip must be set to an obvious, fixed configuration. Only 10Mb/s or 100Mb/s and half **or** full-duplex can be selected.

The contents of the box determines the configuration of the Advertisement Capability Register and defines which communication modes are permitted		Value of the communication speed parameter		
		1	2	3
Value of the duplex mode parameter	1	10Mb/s, half-duplex, fixed	100Mb/s, half-duplex, fixed	not permitted
	2	10Mb/s, full-duplex, fixed	100Mb/s, full-duplex, fixed	not permitted
	3	not permitted	not permitted	not permitted

After the Physical Chip has been set to a fixed mode by the CSF, the PCD7.F65x will not initiate an auto-negotiation cycle with the station at the other end of the cable. It is, however, of the utmost importance that the station at the opposite end has the same configuration as that of the local Physical Chip. Otherwise, stable communications cannot be guaranteed.

#### 4.4.4 CSF SendEtherSBUS

This CSF is like an STXM command (send media). Instead of the IP node, the IP address of the station which will receive the media is specified directly. Sends the source element of the appropriate PCD (Master) to a slave station.

CSF [cc]	S.IP.Library	; IP library
	S.IP.SendEtherSBUS	; "SendEtherSBUS" function
→	Channel	; channel no. (R/K)
→	RemotelP	; remote IP address for sending
		; data (R/K)*
→	SBUSAddr	; remote S-Bus address (R)
→	Count	; number of elements
→	Source	; source I/O/F/R/T/C/DB/K
→	Destination	; destination I/O/F/R/T/C/DB/K
←	Status	; status error message (R)
	* see chapter: IP address decoding	
	Count	1...32
	Number of R/T/C elements sent	1-128
	Number of I/O/F elements sent	
	0	special function code. Please note the information in "Command set for the PCD family" or the PG5 help file
Source	0...8191	Base address of the I/O/F elements in the master PCD
	0...4095	Base address of the R elements in the master PCD
	0...1599	Base address of the T/C elements in the master PCD
	0...7999	Base address of the DB elements in the master PCD
	4000	K, special function code. Please note the information in "Command set for the PCD family" or the PG5 help file
Destination	0...8191	Base address of the I/O/F elements in the slave PCD
	0...4095	Base address of the R elements in the slave PCD
	0...1599	Base address of the T/C elements in the slave PCD
	0...7999	Base address of the DB elements in the slave PCD
	1000	K, write time to the slave PCD
	17, 18, 19	K, special function code. Please note the information in "Command set for the PCD family" or the PG5 help file
Status	0...4095	Status of the CSF in R
		0 = no error
		1 = channel usage error or IP module not present.
		2 = master not assigned
		3 = IP address conversion error
		4 = this type of broadcast is not allowed
		5 = media error



**Example:**

```

$INCLUDE SNetLib.inc      ; include the required SNET library
SASI 9                    ; SASI master, as per the traditional S-Bus, with
                          ; diagnostics
SASI_Master              ; text with diagnostic flags and register
LD R 10                  ; remote IP address is 192.168.12.95
0C0A80C5CH
LD R 20                  ; remote S-Bus address is 95
95

CSF S.SNET.Library
S.SNET.SendEtherSBUS
9                        ; channel 9
R 10                     ; remote IP address
R 20                     ; remote S-Bus address
4                        ; 4 elements
R 0                      ; from registers 0-3 of the master station
R 0                      ; on registers 0-3 on 192.168.12.95, S-Bus 95
R 30                     ; status
    
```



The following table shows which elements from the local source station can be copied to the corresponding elements on the destination station.

		Slave PCD, destination						
		O	F	R	C	T	DB	Clock
Master PCD, Source	I	x	x					
	O	x	x					
	F	x	x					
	R			x	x	x	x	x
	C			x	x	x	x	
	T			x	x	x	x	
	DB			x	x	x		

#### 4.4.5 CSF RecvEtherSBUS

This CSF is like an SRXM command (receive media). Instead of the IP node, the IP address of the station from which the media will be read is specified directly. Reads the source elements from the partner station and loads them into the appropriate PCD over the destination elements.

```
CSF [cc]  S.IP.Library          ; IP library
          S.IP.RecvEtherSBUS ; "RecvEtherSBUS" function
          → Channel            ; channel no. (R/K)
          → RemoteIP          ; remote IP address for reading
          ; data (R/K)*
          → SBUSAddr          ; remote S-Bus address (R)
          → Count             ; number of elements
          → Source            ; source I/O/F/R/T/C/DB/K
          → Destination       ; destination I/O/F/R/T/C/DB/K
          ← Status            ; status error message (R)
```

\* see chapter: IP address decoding

Count	1...32	Number of R/T/C elements sent
	1...128	Number of I/O/F elements sent
	0	special function code. Please note the information in "Command set for the PCD family" or the PG5 help file
	R nnnn	used for data block transfer
Source	0...8191	Base address of the I/O/F elements in the slave PCD
	0...4095	Base address of the R elements in the slave PCD
	0...1599	Base address of the T/C elements in the slave PCD
	0...7999	Base address of the DB elements in the slave PCD
	6000	K, special function code. Please note the information in "Command set for the PCD family" or the PG5 help file
Destination	0...8191	Base address of the I/O/F elements in the master PCD
	0...4095	Base address of the R elements in the master PCD
	0...1599	Base address of the T/C elements in the master PCD
	0...7999	Base address of the DB elements in the master PCD
Status	0...4095	Status of the CSF in R
		0 = no error
		1 = channel usage error or IP module not present.
		2 = master not assigned
		3 = IP address conversion error
		4 = this type of broadcast is not allowed
		5 = media error

#### Example:

```
$INCLUDE SNetLib.inc ; include the required SNET library
  SASI 9             ; SASI master, as per the traditional S-Bus, with
                    ; diagnostics
  SASI_Master       ; text with diagnostic flags and register
  LD R 10           ; remote IP address is 192.168.12.95
  0C0A80C5CH

  LD R 20           ; remote S-Bus address is 95
  95
```

```

CSF  S.SNET.Library
      S.SNET.RecvEtherSBUS
      9                ; channel 9
      R 10             ; remote IP address
      R 20             ; remote S-Bus address
      4                ; 4 elements
      R 0              ; from registers 0-3 of the master station
      R 0              ; on registers 0-3 on 192.168.12.95, S-Bus 95
      R 30             ; status
    
```

The following table shows which elements from the local source station can be copied to the corresponding elements on the destination.

4

Master PCD, destination							
		O	F	R	C	T	DB
Slave PCD, Source	I	x	x				
	O	x	x				
	F	x	x				
	R			x	x	x	x
	C			x	x	x	x
	T			x			x
	K						
	DB			x	x	x	

#### 4.5 Ethernet TCP/IP error messages

Error message	Description:	Action
IPM NOT PRESENT	IP configuration info has been input, but there is no IP module present	Insert an IP module into the slot configured, or delete the IP configuration.
IPM DOES NOT RESTART	The PCD has performed a restart, but the IP module is not responding.	Switch electric supply off, then on.
IPM HAS OLD FW	The IP module firmware is not compatible with the PCD FW.	Update the IP module firmware.
IP FAIL SASITEXT	There is an error in the SASI text	Check SASI text.
IP FAIL SASI DBX	There is an error in the node-list configuration DBX	Check node configuration Attention: there must be at least one IP node in the node list. Configure the IP module and insert it into the PCD.
IP FAIL NO IPM	An IP function has been executed; however, no IP configuration or IP module is present.	

## 5 Diagnostics and troubleshooting

### 5.1 Summary

There are two main phases to all troubleshooting measures, the diagnostic phase and the solution implementation phase. Analysis of the problem involves gradually narrowing down the potential causes of an error by asking questions intended to eliminate certain causes. It is important to locate the cause of all errors.

### 5.2 The TCP/IP communication process

5

The communication process between two PCD stations on a network can be subdivided into three stages. When a telegram is sent from one PCD to another, it goes through these stages in the following order.

- Conversion of the IP-node identifier into the local IP address (configured in the Hardware Settings).
- Conversion of the transfer IP address into a MAC address (using the ARP table). When broadcast messages are sent, the MAC address is always used in the following format:

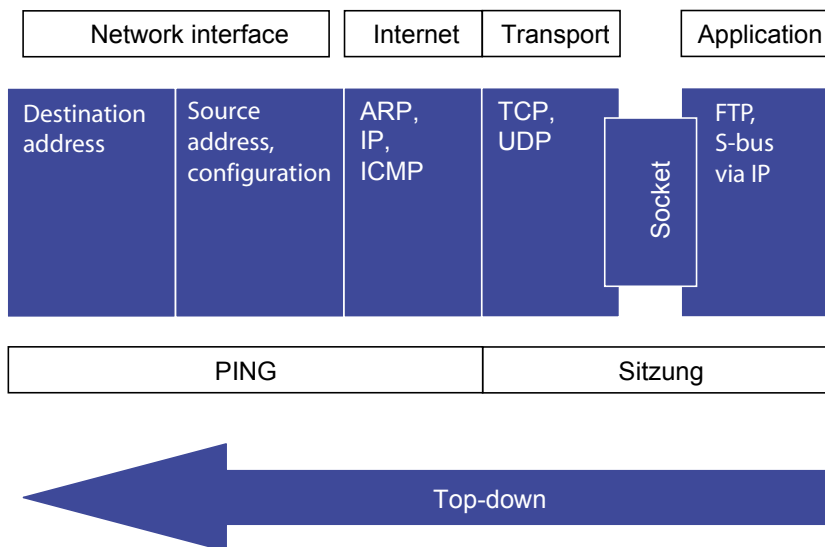
0xFF-FF-FF-FF-FF-FF

- Transfer of the IP datagram to the MAC address provided by the ARP.

Each TCP/IP stack follows this sequence in order to send a telegram from one point to another.

### 5.3 The top-down rule

When performing troubleshooting for TCP/IP, we recommend starting with the top level (application layer) and gradually proceeding downwards to the lower IP layers. Troubleshooting includes checking whether the protocols for each layer are able to communicate with the layers above and below.



After the Ethernet TCP/IP module has been configured in the PG5 Hardware Settings - IP address, subnet mask, default gateway, station number (S-Bus address) and IP node - troubleshooting proceeds in two stages:

- 1) Establish a connection between a PG5 Client and a PCD Slave.

To connect the PG5 Online Debugger:

- Select SOCKET as the IP Channel Type in the PG5 Online Settings.
- Modify the settings (if necessary)
- Bring the PG5 Online Debugger online.

If this works, a connection will be established between the network interface layer and the application layer. All the protocols between the two layers are working properly.

If it does not work, check the connection between the network interface layer and the internet layer.

- 2) Follow the top-down rule and use the PING command to check that the ARP is working properly, i.e. whether the IP address is converted into the right MAC address for each echo request and each response to an echo request.

### 5.3.1 PING

If a PING command (echo request) is successful, this means that TCP/IP is installed correctly on the host computer and on the Ethernet TCP/IP module, and that the ARP cache list contains the PCD stations.

Examples:

PING 127.0.0.1	“Look up address”: checks that the TCP/IP stack on the host computer is properly installed. The echo request remains in the local TCP/IP stack.
PING 192.168.12.60	Own IP address: checks that the station is listed correctly in the ARP cache table (used for converting the IP address to the MAC address).
PING 192.168.12.60	IP address of the remote PCD unit: checks that the TCP/IP stack (including the ARP cache) is properly installed on the remote PCD station.
PING 192.168.12.60	The IP address of a PCD station can only be obtained via a router: Checks that the configured default gateway address matches.

PING <IP-Address> -t	Continually transmits echo requests
PING <IP-Address> -n 10	Transmits 10 consecutive echo requests
PING <IP-Address> -L 320	Transmits a request of 320 bytes in length

```
C:\WINNT\system32>ping 192.168.12.60
Pinging 192.168.12.60 with 32 bytes of data:
Reply from 192.168.12.60: bytes=32 time<10ms TTL=30
Reply from 192.168.12.60: bytes=32 time<10ms TTL=30
Reply from 192.168.12.60: bytes=32 time<10ms TTL=30
Reply from 192.168.12.60: bytes=32 time<10ms TTL=30
```

Time	Reaction time for the echo request
TTL	Time-to-live (validity period): this number decreases with each pass through a switch or router.  When the TTL reaches zero, no more telegrams will be sent.

If a PING command does not work, check the ARP cache list in order to determine whether it contains the PCD station and if address conversion is working (IP address to MAC address).

**5.3.2 ARP**

The ARP command lists the contents of the ARP cache table on the local host computer. An ARP entry in the cache table contains the remote IP and MAC addresses.

If two stations on the same subnet are unable to exchange PING requests, check that the station is listed in the ARP cache table.

ARP -a	Displays all the entries in the ARP cache
ARP -d <IP-Address>	Deletes an entry from the ARP cache

```
C:\WINNT\system32>arp -a

Interface: 192.168.12.46 on Interface 2
Internet Address      Physical Address      Type
192.168.12.60         00-50-c2-0c-50-24    dynamic
192.168.12.200        00-10-83-f9-1d-fe    dynamic
192.168.12.202        00-30-6e-00-03-50    dynamic
192.168.12.221        00-80-3e-7f-15-fa    dynamic
```

**5.3.3 Other useful commands for the host computer**

IPCONFIG Displays the TCP/IP configuration parameter for the host computer

```
C:\WINNT\system32>ipconfig

Windows NT IP Configuration

Ethernet adapter E100B1:

    IP Address. . . . . : 192.168.12.46
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.12.221

C:\WINNT\system32>
```

```
C:\WINNT\system32>ipconfig/all

Windows NT IP Configuration

    Host Name . . . . . : ch01w234.saia-burgess.ch
    DNS Servers . . . . . : 192.168.6.204
                        192.168.6.28
    Node Type . . . . . : Hybrid
    NetBIOS Scope ID. . . . . :
    IP Routing Enabled. . . . . : No
    WINS Proxy Enabled. . . . . : No
    NetBIOS Resolution Uses DNS : No

Ethernet adapter E100B1:

    Description . . . . . : Intel(R) PRO Adapter
    Physical Address. . . . . : 00-90-27-BC-ED-8A
    DHCP Enabled. . . . . : No
    IP Address. . . . . : 192.168.12.46
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.12.221
    Primary WINS Server . . . . . : 192.168.6.204
    Secondary WINS Server . . . . . : 192.168.6.209
```

**NETSTAT** Displays the statistics and protocols for existing TCP/IP connections on the host computer

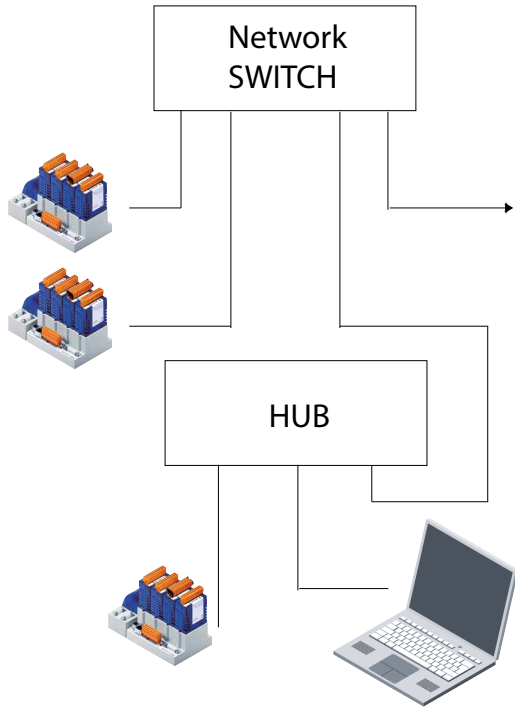
NETSTAT -a	Displays a list of all active connections and ports
NETSTAT -r	Lists the contents of the routing table
NETSTAT -p UDP	Displays a list of all active UDP connections
NETSTAT -p TCP	Displays a list of all active TCP connections
NETSTAT -p ICMP	Displays a list of all active ICMP connections

**5.4 Wireshark Ethernet protocol analyzer**

We recommend the freeware tool Wireshark for analysing proprietary Ether-S-Sus communication and general Ethernet communication ([www.wireshark.org](http://www.wireshark.org)).

This is an open-source analyzer, which includes the Ethernet card logger WinPcap. Ethernet recordings made using this program can be analysed and interpreted. Wireshark Analyzer, Version 1.1 and later, can also interpret Ether-S-Bus telegrams. Please see the relevant FAQ (#100569) for more detailed information on this topic.

In order to record all communication from/to the station in question, a hub should be connected ahead of it. The analyzer then listens to this hub. Switches route the communications traffic into channels and are not suitable for analyses and recordings.



5

This is an interpreted S-Bus UDP telegram (red flags) recorded in Wireshark

The screenshot shows the Wireshark interface. The packet list pane on the left contains a table of captured packets. The packet details pane on the right shows the structure of a selected S-Bus UDP telegram.

No.	Time	Source	Destination	Protocol	Info
867	9.870	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
868	9.878	172.16.1.142	172.16.1.140	S-Bus	Response
869	9.890	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
870	9.898	172.16.1.142	172.16.1.140	S-Bus	Response
871	9.909	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
872	9.918	172.16.1.142	172.16.1.140	S-Bus	Response
873	9.939	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
874	9.948	172.16.1.142	172.16.1.140	S-Bus	Response
875	9.959	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
876	9.968	172.16.1.142	172.16.1.140	S-Bus	Response
877	9.969	172.16.1.140	172.16.1.142	S-Bus	Request: Read PCD status (own)
878	9.978	172.16.1.142	172.16.1.140	S-Bus	Response
879	9.979	172.16.1.140	172.16.1.142	S-Bus	Request: Read instruction point
880	9.988	172.16.1.142	172.16.1.140	S-Bus	Response
881	9.999	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
882	10.008	172.16.1.142	172.16.1.140	S-Bus	Response
883	10.030	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
884	10.038	172.16.1.142	172.16.1.140	S-Bus	Response
885	10.049	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
886	10.058	172.16.1.142	172.16.1.140	S-Bus	Response
887	10.069	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
888	10.078	172.16.1.142	172.16.1.140	S-Bus	Response
889	10.100	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
890	10.108	172.16.1.142	172.16.1.140	S-Bus	Response
891	10.119	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
892	10.128	172.16.1.142	172.16.1.140	S-Bus	Response
893	10.139	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
894	10.148	172.16.1.142	172.16.1.140	S-Bus	Response
895	10.170	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
896	10.178	172.16.1.142	172.16.1.140	S-Bus	Response
897	10.189	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
898	10.198	172.16.1.142	172.16.1.140	S-Bus	Response
899	10.209	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
900	10.218	172.16.1.142	172.16.1.140	S-Bus	Response
901	10.240	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
902	10.248	172.16.1.142	172.16.1.140	S-Bus	Response
903	10.259	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
904	10.268	172.16.1.142	172.16.1.140	S-Bus	Response
905	10.279	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
906	10.290	172.16.1.142	172.16.1.140	S-Bus	Response
907	10.311	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
908	10.318	172.16.1.142	172.16.1.140	S-Bus	Response
909	10.329	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
910	10.338	172.16.1.142	172.16.1.140	S-Bus	Response
911	10.349	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
912	10.358	172.16.1.142	172.16.1.140	S-Bus	Response
913	10.380	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
914	10.388	172.16.1.142	172.16.1.140	S-Bus	Response
915	10.399	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
916	10.408	172.16.1.142	172.16.1.140	S-Bus	Response
917	10.419	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
918	10.428	172.16.1.142	172.16.1.140	S-Bus	Response
919	10.450	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
920	10.458	172.16.1.142	172.16.1.140	S-Bus	Response
921	10.469	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
922	10.478	172.16.1.142	172.16.1.140	S-Bus	Response
923	10.489	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
924	10.498	172.16.1.142	172.16.1.140	S-Bus	Response
925	10.520	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte
926	10.528	172.16.1.142	172.16.1.140	S-Bus	Response
927	10.539	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
928	10.548	172.16.1.142	172.16.1.140	S-Bus	Response
929	10.559	172.16.1.140	172.16.1.142	S-Bus	Request: Read flag(s)
930	10.568	172.16.1.142	172.16.1.140	S-Bus	Response
931	10.590	172.16.1.140	172.16.1.142	S-Bus	Request: Read byte

SAIA S-Bus

Ether-S-Bus header

- Telegram attribute: Request (0x00)
- Destination: 142
- Command: Read Flag(s) (0x02)
- R-count: 8
- Base address IOF: 0
- Checksum: 0xdcdc (correct)

```
0000 00 50 c2 4b c6 b9 00 08 c7 f9 b5 ba 08 00 45 00 .P.K.....
0010 00 3c 11 90 00 00 80 11 cd f6 ac 10 01 8c ac 10 .....
0020 01 8e 06 cc 13 ba 00 18 77 14 80 00 00 10 01 00 ..... w...
0030 72 67 00 8e 02 07 00 00 dc 00 00 00 ..... d...
```



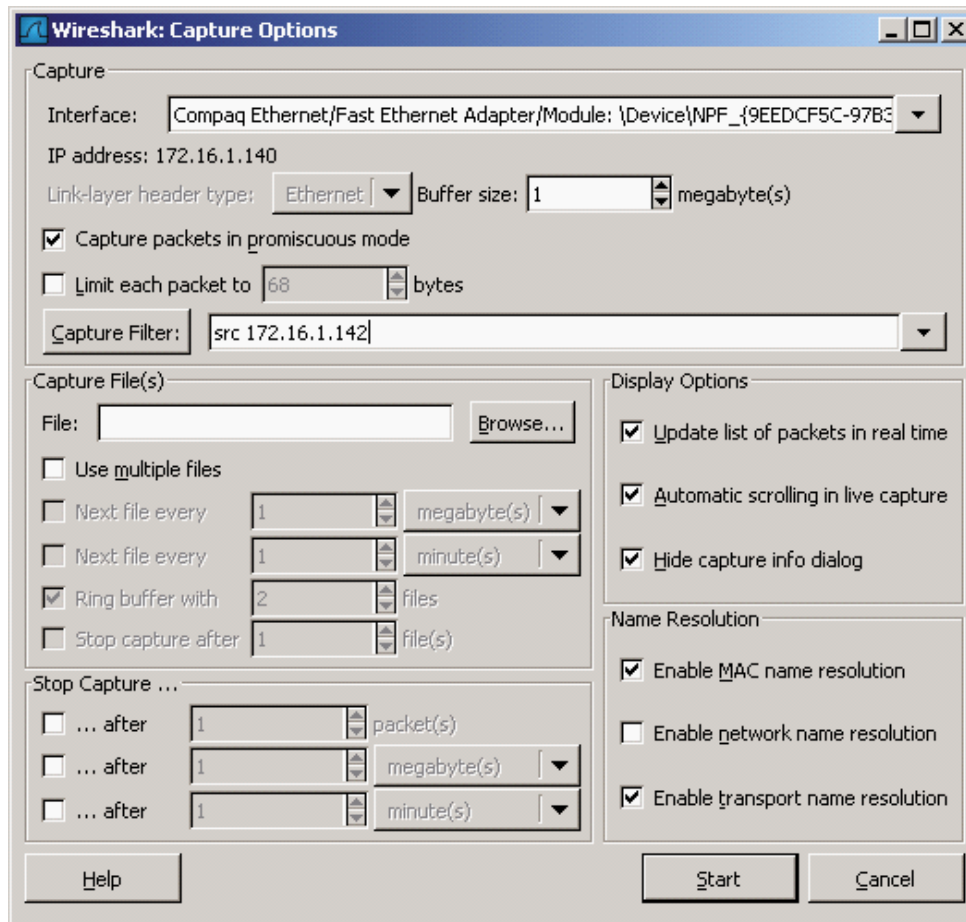
Interpretations of other protocols, such as ModBus TCP or ModBus UDP, can also be displayed.

Wireshark offers very useful filter functions, which can either be applied while data logging is taking place, or later during analysis. It is often advisable to record the communication without “Capture Filter” selected and to apply the filter later during analysis, using “Display Filter”.

Here are a few examples of this:

“**Capture Filters**” - only those telegrams which meet the filter conditions are recorded. The Capture Filter can be accessed via “Capture → Options...” or using the shortcut Ctrl+K. Using the settings shown in the figure below, all telegrams to/from IP address 172.16.1.142 - whether incoming or outgoing - are recorded.

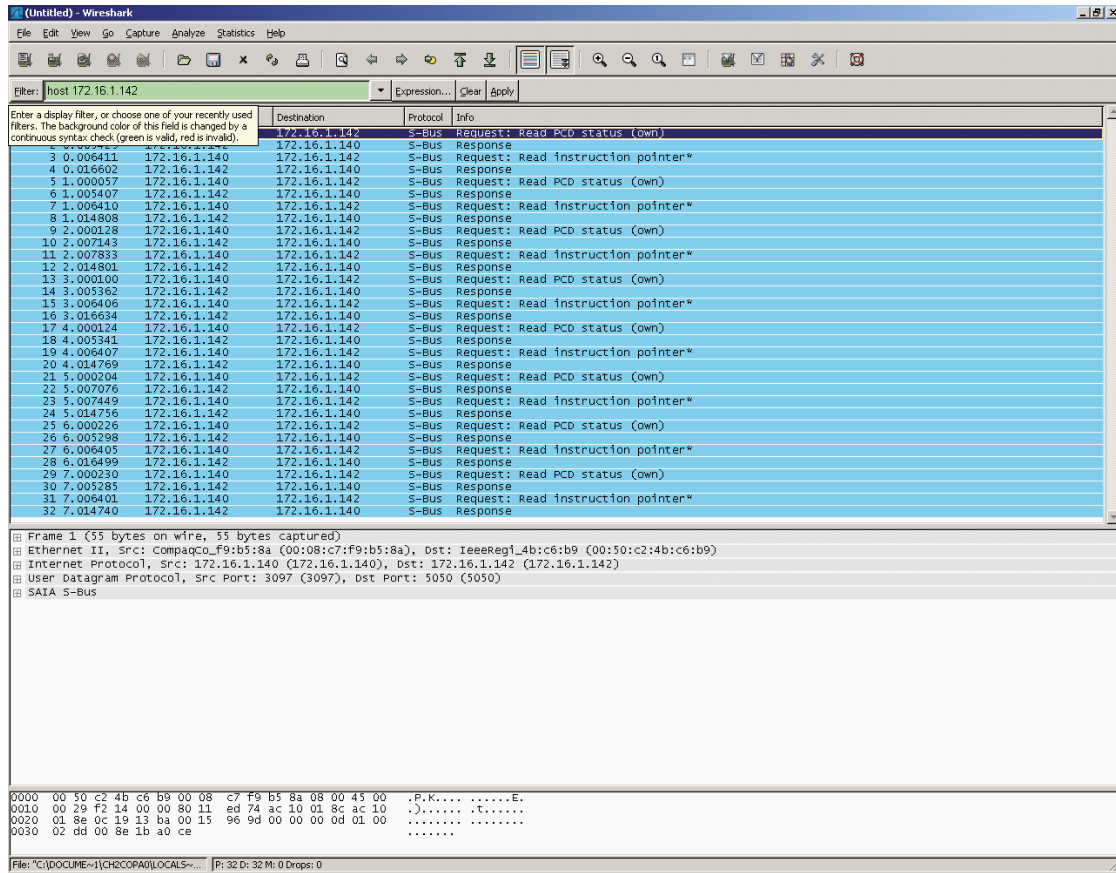
5



Examples of other recording filters are:

Host 172.18.5.4	only records the traffic from or to an IP address 172.18.5.4
Port 5050	only records the traffic using port 5050
SRC 172.18.5.4	records all outgoing telegrams from this IP address
DST port 135 and TCP port 135	records the traffic on destination port 135 in TCP

“Display Filters” - these filters relate only to the display and do not change the contents of the log file. They are sorting filters. The display filter is defined right in the main window.



5

Examples of display filters:

- Udp displays all UDP traffic
- Tcp displays all TCP traffic
- Host 172.18.5.4 displays the traffic for IP address 172.18.5.4
- Host 172.18.5.4 and port 5050 displays the traffic for IP address 172.18.5.4 using port 5050

Functions using relational operators:

- tp.addr == 172.18.5.4 displays the traffic for IP address 172.18.5.4
- tp.addr != 172.18.5.4 displays all traffic except that via IP address 172.18.5.4
- !(ip.addr == 172.18.5.4) displays all traffic except that via IP address 172.18.5.4
- ip.src == 172.18.5.4 and ip.dst == 172.18.5.5 displays all telegrams from 172.18.5.4 to 172.18.5.5
- Frame.pkt\_len < 128 displays all packets smaller than 128 bytes
- Tcp.port == 25 or icmp displays all telegrams via TCP port 25 and ICMP telegrams
- Tcp.window\_size == 0 && tcp.flags.reset != 1 TCP window size is zero Buffer is full

The following comparisons are permitted:

eq	==	and	&&
ne	!=	or	
gt	>	xor	^^
lt	<	not	!
ge	>=		
le	<=		

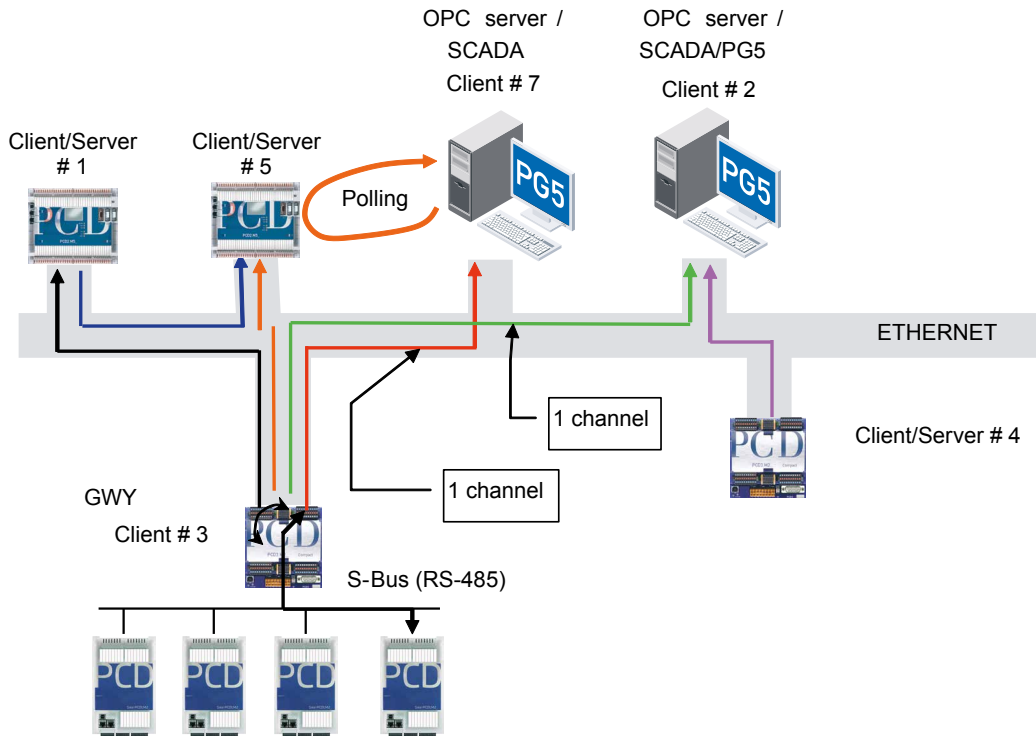
Boolean operators, for instance for the six TCP control bits, are not permitted. For example, the condition: tcp.flags.syn

URG: Urgent Pointer field significant

ACK: Acknowledgement field significant

PSH: Push function (send data from the stack)

RST: Reset the connection (e.g. the response if a telegram goes to a port which is not open)



SYN: Synchronize sequence numbers (start of a TCP connection)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.12.92	Broadcast	ARP	who has 192.168.12.93? Tell 192.168.12.92
2	0.000408	IeeeRegI_Oc:54:0c	192.168.12.92	ARP	192.168.12.93 is at 00:50:c2:0c:54:0c
3	0.001583	192.168.12.92	192.168.12.93	TCP	6000 > 6000 [SYN] Seq=769525 Ack=0 Win=720 Len=0 MSS=1458
4	0.002373	192.168.12.93	192.168.12.92	TCP	6000 > 6000 [SYN, ACK] Seq=384001 Ack=769526 Win=720 Len=0 MSS=1458
5	0.004847	192.168.12.92	192.168.12.93	TCP	6000 > 6000 [ACK] Seq=769526 Ack=384002 Win=720 Len=0

FIN: No more data from sender (end of TCP connection)

No. -	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.12.92	192.168.12.93	TCP	6000 > 6000 [FIN, ACK] Seq=833653 Ack=768002 win=720 Len=0
2	0.000609	192.168.12.93	192.168.12.92	TCP	6000 > 6000 [ACK] Seq=768002 Ack=833654 win=720 Len=0
3	0.001954	192.168.12.93	192.168.12.92	TCP	6000 > 6000 [FIN, ACK] Seq=768002 Ack=833654 win=720 Len=0
4	0.003014	192.168.12.92	192.168.12.93	TCP	6000 > 6000 [ACK] Seq=833654 Ack=768003 win=720 Len=0

FAQ 100535 provides further information about filter functions.

**5.5 PCD7.F655 IP stack debugging via RS-232 on the Saia PCD®**

The TCP/IP stack of a PCD7.F655 can be debugged via the RS-232 serial interface on the PCD. For example, statistics on Ethernet traffic can be analysed, the ARP table for the PCD7.F655 can be viewed, or the open sockets can be monitored. Details on the various functions are listed below.

The PCD7.F655 can be debugged via the following PCD ports:

PCD...	PCD7.F655 and up...	Debug via...
PCD2.M150	Slot B1	Port 2 (Pins 30-34)
PCD2.M170	Slot B2	Port 4 (Pins 40-44)
PCD4.M170	Slot B2	Port 4 (Pins 40-44)
PCD2.M480	Slot B1, Slot B2	Port 2 (Pins 30-34) Port 4 (Pins 40-44)



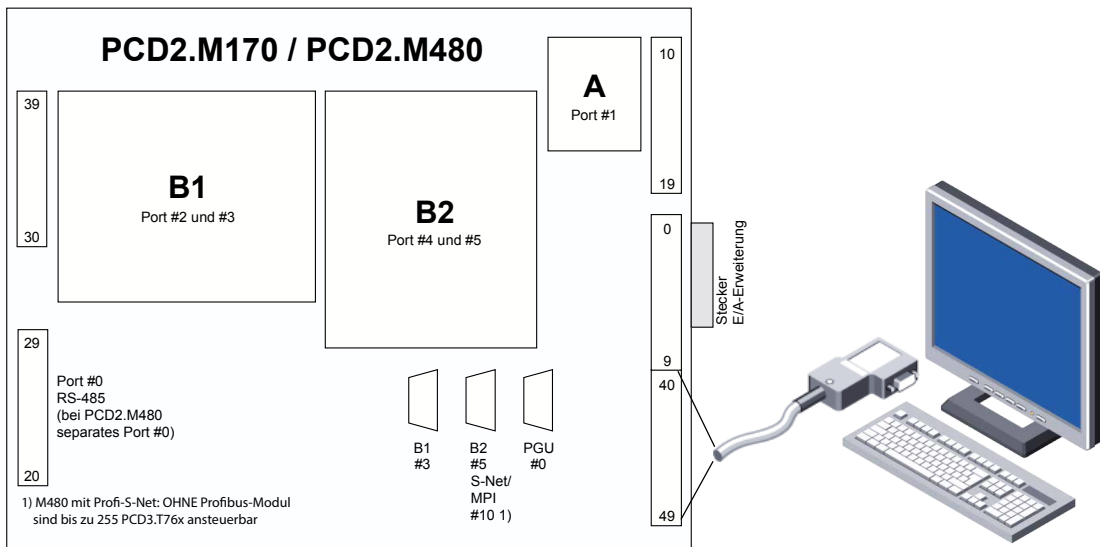
The PCD7.F655 on a PCD1.M13x cannot be debugged, because it does not have a serial interface.

Connect the serial interface on the PCD to your PC with a null-modem cable (only GND, TxD and RxD are used) and open, for example, a HyperTerminal connection with the following parameters:

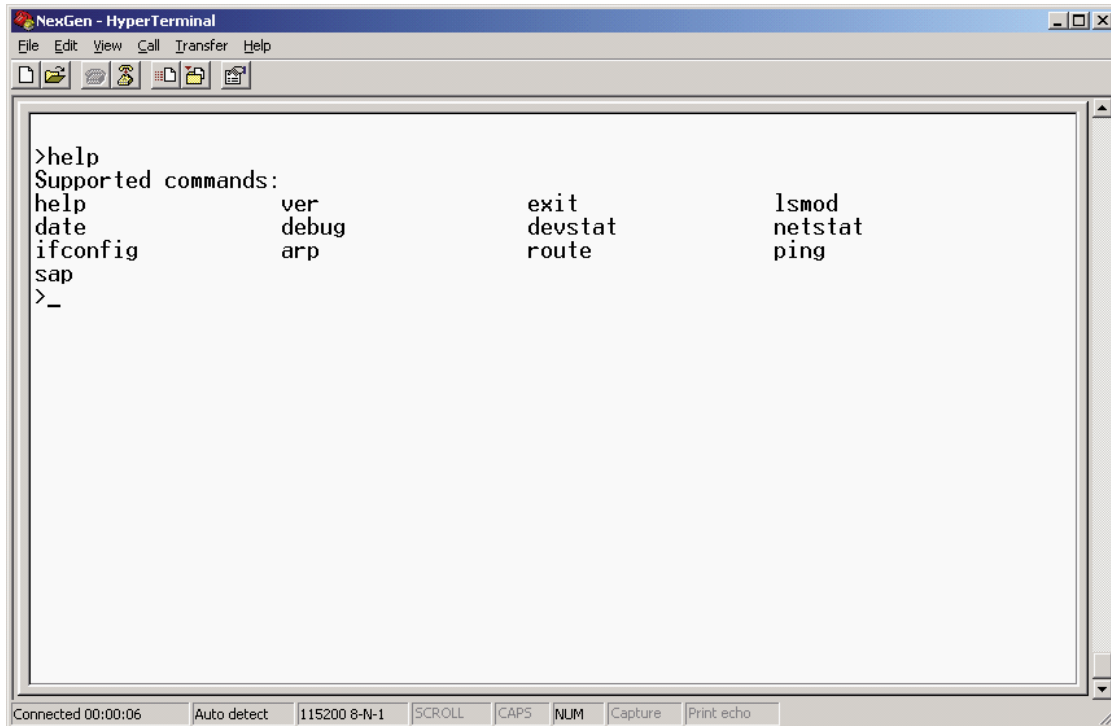
- Bits per second: 115'200
- Data Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

After pressing the Enter key, the “>” prompt appears in the shell. You can now issue the debug command via the shell.

In the following example, the PCD7.F655 is inserted in slot B2 of a PCD2.M170. This being the case, you should connect port 4 (pins 40-44) on the PCD2.M170 to your PC to start the communication shell.



When the shell starts, you can use “help” to list all functions supported for debugging the TCP/IP stack.



```

NexGen - HyperTerminal
File Edit View Call Transfer Help
>help
Supported commands:
help          ver          exit          lsmmod
date          debug        devstat       netstat
ifconfig      arp          route         ping
sap
>_
Connected 00:00:06  Auto detect  115200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo

```

5

- “**ver**”            Stack version
- “**netstat**”        Active network connections
  - a
  - s        Precise statistics about IP, ICMP, IGMP, TCP and UDP telegrams. Missing telegrams can be indentified here.
  - b        Manage the available memory on the PCD7.F655
- “**ifconfig**”       Ethernet interface for the station
- “**arp**”            ARP table for the station
- “**ping**”           Ping the remote station
- “**sap**”            Sockets which are open on the module
  - PCD1: **always** Sap 16 for S-Bus UDP port 5050
  - For all other PCDs which support IP, always Sap 32 for S-Bus UDP port 5050
- help <Command>** Displays the possible call arguments for the function.
  - e.g. “>help netstat”

Below are some examples of using the debug function while communication is in progress.

The shell command “**ifconfig**” displays the Ethernet interface of the PCD7.F655. In this case, the IP address is 172.16.1.142, the subnet 172.16.1.0, and the broadcast address for the configured network is 172.16.1.255.

The MAC address is also visible (00 50 C2 4B C6 BC), along with rough statistics on the number of ingoing and outgoing telegrams.

```

NexGen - HyperTerminal
File Edit View Call Transfer Help
>ifconfig
Network Interfaces
lo0    Local Loopback
  Addr:127.0.0.1 SubNet:127.0.0.0 Bcast:127.255.255.255
  UP BROADCAST RUNNING MTU:1504
  RX Pkts:0 Mcast:0 Bytes:0 Errs:0 Drops:0 NoProto:0
  TX Pkts:0 Mcast:0 Bytes:0 Errs:0 Drops:0
  Driver:LOOPBACK
eth0   Ethernet      HWaddr:0050C2-4BC6BC
  Addr:172.16.1.142 SubNet:172.16.1.0 Bcast:172.16.1.255
  UP BROADCAST RUNNING MULTICAST MTU:1500
  RX Pkts:2252 Mcast:20 Bytes:104846 Errs:0 Drops:0 NoProto:0
  TX Pkts:2218 Mcast:1 Bytes:138953 Errs:0 Drops:0
  Driver:NET-ARM Irq:0 IOBase:0x0 Mem:0x00000000 PHY:PCDPHY
  Link status: 10Base-T
  Multicast Filter:
    01005E-000001
>_
Connected 00:00:10  Auto detect  115200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
    
```

The shell command “**sap**” displays the node points (Saps) which are open on the PCD7.F655. UDP port 5050 (S-Bus UDP9 is always Sap 32 (except PCD1.M13x, where it is Sap 16). A TCP Client connection is opened at Sap 10 on station 172.16.1.141 at the remote port 5050. The status is “connected”. This also shows that UDP communication on port 5050 and TCP communication on port 5050 are working simultaneously.

The shell command “**arp**” displays the internal ARP table for the local station. In the example, the PC running the PG5 Online Debugger (172.16.1.140) and the remote station (172.16.1.141) - with which there is a TCP connection on port 5050 - are shown.

The shell command “**netstat**” displays the different sockets which are open on the local station. This is **always** the UDP socket on port 5050 (S-Bus UDP) and the TCP socket on port 1700 (FTP Server socket for the PCD7.F655 firmware update). In the example, the previously addressed TCP Client socket is still open.

```

NexGen - HyperTerminal
File Edit View Call Transfer Help
>
>
>
>
>
>sap
SAP Entries:
  32: fastMbx, udp, noFrame, port=5050, timeo=-1
     sock 00067E70: 0.0.0.0:0, udp, idle
  10: slowMbx, tcpClient, noFrame, port=5050, timeo=-1
     sock 00067EAC: 172.16.1.141:5050, connected, idle
>
>arp
Address      HWAddress    Flags  Timer
172.16.1.140 0008C7-F9B58A ---    38
172.16.1.141 0050C2-0C5F1F ---    49
>
>netstat
Active Internet connections
Proto Recv-Q Send-Q Local Address Foreign Address State
udp    0      0          *:5050          *:*
tcp    0      0          *:1700          *:* LISTEN
tcp    0      0 172.16.1.142:5050 172.16.1.141:5050 ESTABLISHED
>_
Connected 00:01:02  Auto detect  115200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
    
```

The shell command “**netstat -s**” displays precise statistics about IP, ICMP, IGMP, TCP, and UDP telegrams. Missing telegrams can be indentified here.

```
>netstat -s
```

```
Ip:
```

```
8424 total packets received
0 with invalid header
0 with invalid address
0 forwarded
0 with unknown protocol
0 incoming packets discarded
8424 incoming packets delivered
8390 requests sent out
0 outgoing packets dropped
0 dropped because of missing route
0 reassemblies required
0 packets reassembled ok
0 packets reassembles failed
0 packets fragmented ok
0 fragments failed
0 fragments created
```

```
Icmp:
```

```
0 messages received
0 messages received with error
Input histogram:
0 messages sent
0 messages not sent
Output histogram:
```

```
Igmp:
```

```
0 messages received
0 messages received with error
0 membership queries received
0 membership reports received
0 membership reports received for our groups
0 membership reports sent
```

```
Tcp:
```

```
1 active opens
0 passive opens
0 embryonic connections dropped
0 established connections dropped
10 packets sent (0 retransmitted)
0 RESET packets sent
9 packets received
0 packets received with errors
0 duplicate ACKs
0 out-of-order packets
```

```
Udp:
```

```
8400 packets received
35 packets to unkown port received
0 packets received with errors
8380 packets sent
```



Example of a real application, consisting of three PCDs (local 192.100.100.106 and remotes 192.100.100.104 and 192.100.100.100) which communicate with one another in S-Bus UDP (port 5050). Recording takes about one week.

```
>sap
SAP Entries:
    32:   fastMbx, udp, noFrame, port=5050, timeo=-1
        sock 000680C8: 0.0.0.0:0, udp, idle

>arp
Address          HWAddress          Flags          Timer
192.100.100.104  0050C2-4BC21D      ---           119
192.100.100.100  000D56-941547      ---           120

>ifconfig
Network Interfaces
lo0  Local Loopback
    Addr:127.0.0.1 SubNet:127.0.0.0 Bcast:127.255.255.255
    UP BROADCAST RUNNING MTU:1504
    RX Pkts:0 Mcast:0 Bytes:0 Errs:0 Drops:0 NoProto:0
    TX Pkts:0 Mcast:0 Bytes:0 Errs:0 Drops:0
    Driver:LOOPBACK

eth0      Ethernet          HWaddr:0050C2-4BC6B4
    Addr:192.100.100.106 SubNet:192.100.100.0
    Bcast:192.100.100.255
    UP BROADCAST RUNNING MULTICAST MTU:1500
    RX Pkts:52921868 Mcast:81423 Bytes:2723134990 Errs:39 Drops:0
    NoProto:0
    TX Pkts:52846356 Mcast:15481 Bytes:2016432650 Errs:2 Drops:0
    Driver:NET+ARM lrq:0 IOBase:0x0 Mem:0x00000000 PHY:PCDPHY
    Link Status:  lo0Base-TX
    Multicast Filter:
        01005E-000001

>netstat -b
Network message buffers
total:   126 free:   116, 15560/196056 bytes in use
Socket control blocks
total:   200 free:   198, 336/33600 bytes in use
TCP control blocks
total:   200 free:   199, 212/42400 bytes in use

>netstat -s
Ip:
    52851798 total packets received
    28 with invalid header
    207 with invalid address
    0 forwarded
    0 with unknown protocol
    0 incoming packets discarded
    52851563 incoming packets delivered
    53295668 requests sent out
    0 outgoing packets dropped
    0 dropped because of missing route
    0 reassemblies required
    0 packets reassembled ok
    0 packets reassembles failed
    0 packets fragmented ok
    0 fragments failed
    0 fragments created
```



## Icmp:

```
191684 messages received
0 messages received with error
Input histogram:
  echo reply: 4
  echo request: 191680
191684 messages sent
0 messages not sent
Output histogram:
  echo reply: 191680
  echo request: 4
```

## Igmpp:

```
0 messages received
0 messages received with error
0 membership queries received
0 membership reports received
0 membership reports received for our groups
0 membership reports sent
```

## Tcp:

```
0 active opens
0 passive opens
0 embryonic connections dropped
0 established connections dropped
0 packets sent (0 retransmitted)
0 RESET packets sent
0 packets received
0 packets received with errors
0 duplicate ACKs
0 out-of-order packets
```

## Udp:

```
52659652 packets received
25391 packets to unknown port received
0 packets received with errors
53103984 packets sent
```

## 6 Programming examples

### 6.1 Reference to Ethernet links

We have done away with any references to Ethernet literature; such a huge number of books have been written on this topic that picking out just a few serves no purpose. There is also a great deal of information to be found on the internet.

**IANA** (International Assigned Numbers Authority). Full text search of the RFC Collection, etc. <http://www.iana.org>

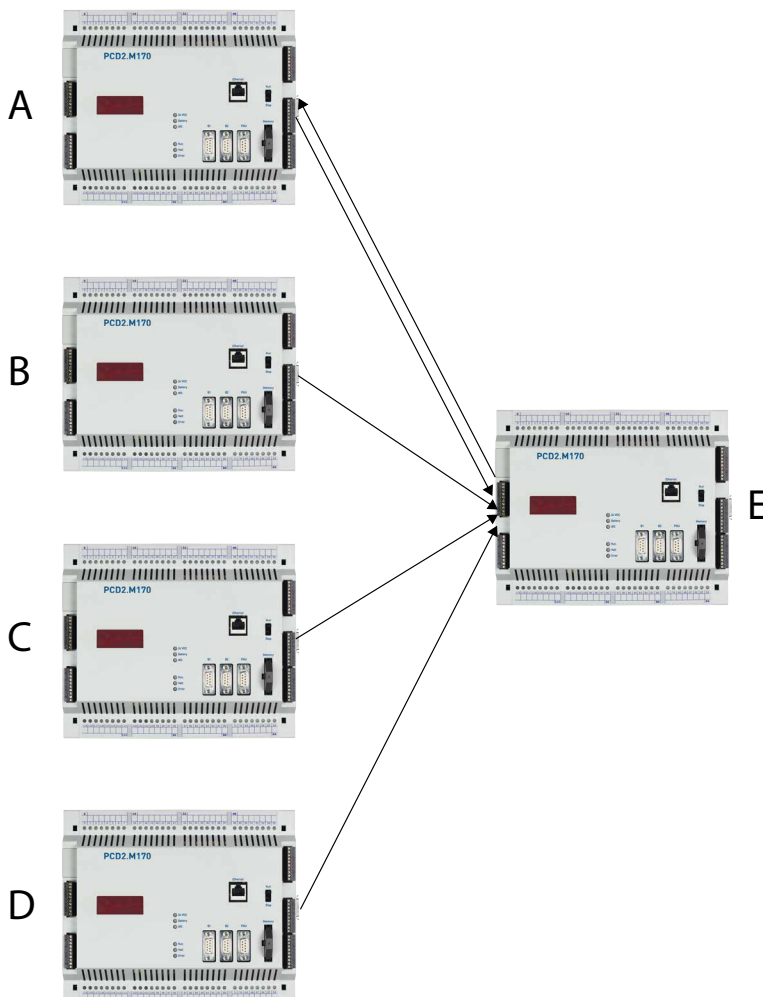
**Hirschmann:** For our internal tests, we mostly use industrial equipment made by Hirschmann and have good experiences with this supplier. Their home page, with an excellent full-text search and good manuals, can be found at: <http://www.hirschmann.com>

A free **Ethernet Analyzer** can be found at <http://www.wireshark.org>. For more details, see FAQ #100569.

6

### 6.2 Performance testing

The results are based on measurements on a separate, local network with a hub. PCD2.M170 was used, and the PCD7.F65x was equipped with firmware version 041. Testing was performed with S-Bus UDP communication.



**Saia PCD® E stopped**

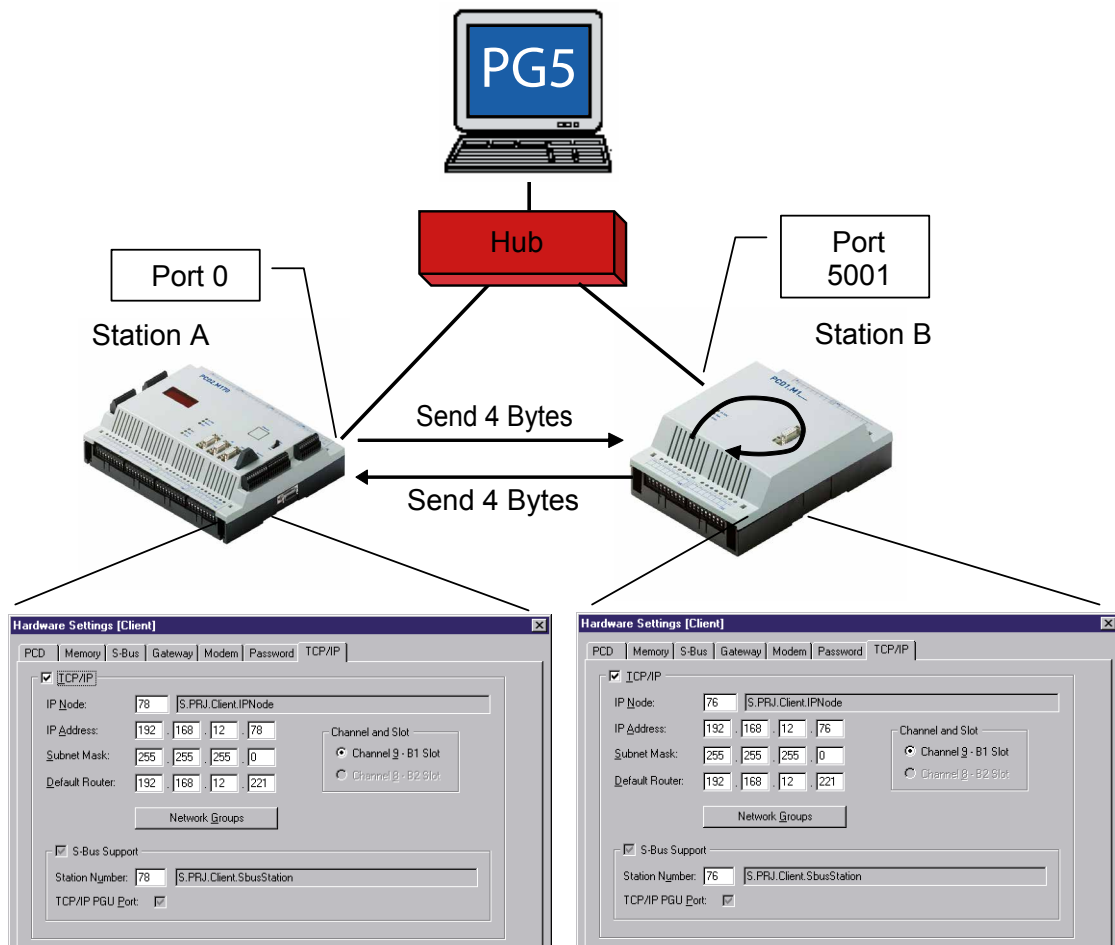
A → E	A: 124 S-Bus tlgs/s i.e. 1 S-Bus telegram requires 8 ms
A → E + B → E	A + B: 112 + 112 = 224 S-Bus tlgs/s
A → E + B → E + C → E	A + B + C: 103 + 103 + 103 = 309 S-Bus tlgs/s
A → E + B → E + C → E + D → E	A + B + C + D: 86 + 85 + 85 + 86 = 342 S-Bus tlgs/s This configuration requires nearly the full bandwidth

**Saia PCD® E sends S-Bus telegrams to Saia PCD® A**

A → E + B → E + C → E + E → A	E: 79 S-Bus Tlgs/s
A → E + B → E + C → E + D → E + E → A	E: 63 S-Bus Tlgs/s

**6.3 Example program: TCP/IP Open Data Mode**

The following is a programmed example of a TCP/IP connection in Open Data Mode. It implements a client echo server in TCP/IP. A previously initialised register is sent to the server, which then sends it back. The client increments the register and sends the next request.

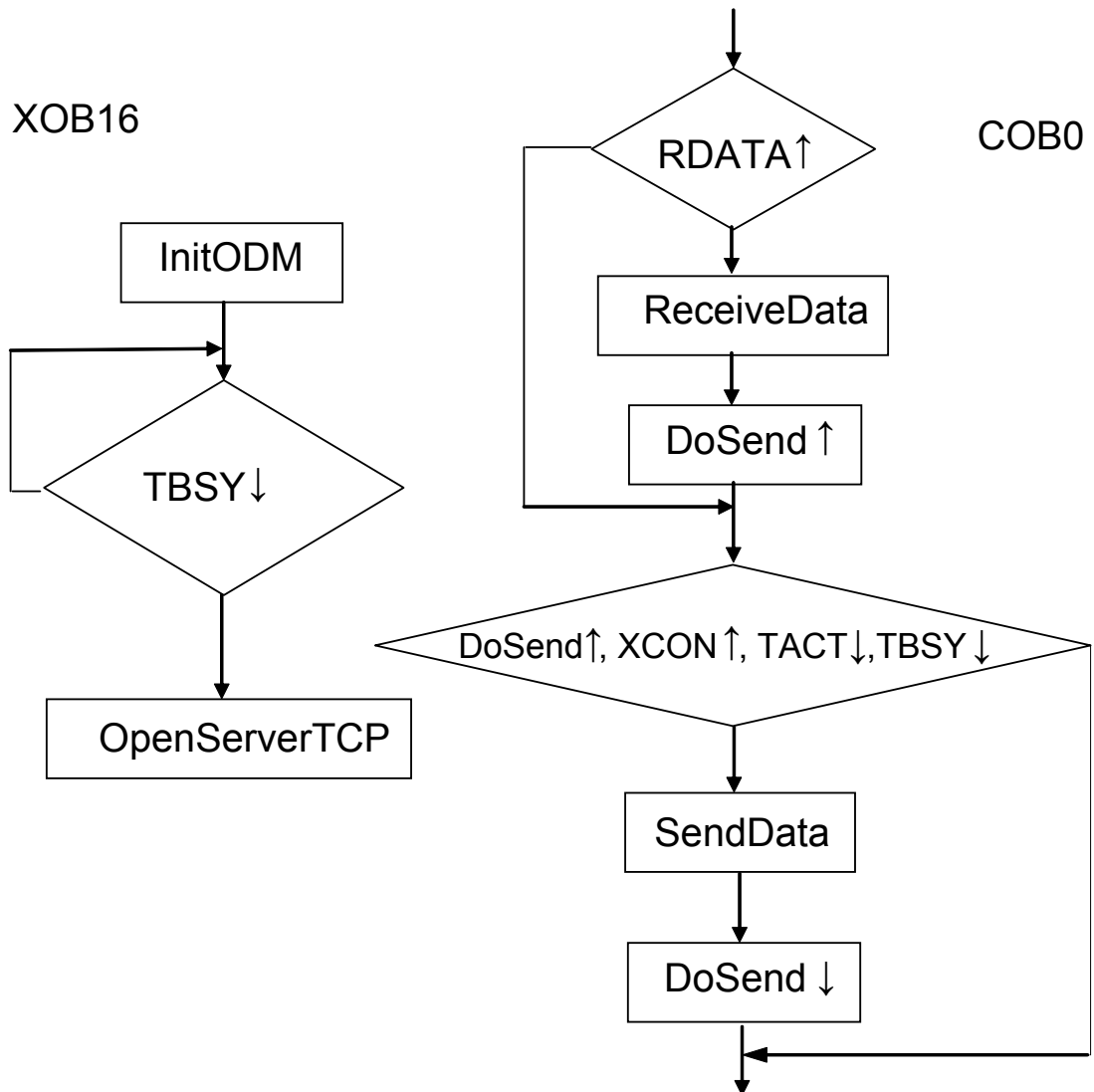


**6.3.1 Server**

**Flow diagram of the server**

- XOB 16 Initialises Open Data Mode  
Opens a TCP Server socket on port 5001  
Switches to “automatic client acceptance” mode.  
Waits for the connection to the client.
- COB 0 Receives data  
Sends the received data back

As an aid to clear presentation, “JR-1” steps were used for the YES/NO queries.



**Server code listing (IL)**

“JR-1” has been used for ease of presentation.

```

#include "IPLib.inc"

$init

    LD    Conn_Tout          ;load timeout as unlimited
        0

    CSF   S.IP.Library      ;initialise Open Data Mode
        S.IP.InitODM
        GlobalFlags        ;global diag. flags
        ChannelReg         ;for multiple channels:
                            ;this is where you will find the
                            ;number of the channel in which the
                            ;received data has been written
        0                   ;timeout 0 means: remains
                            ;unlimited in mbx

    STL   GlobalFlags[TBSY] ;check if the transmitter is free
    JR    L-1

                            ;open the TCP Server port

    CSF   S.IP.Library
        S.IP.OpenServerTCP
        Channel             ;the virtual channel on which data
                            ;is sent and received
        Local_Port         ;the local IP port over which the
                            ;data is sent and received
        DiagFlag           ;channel diagnostic flags
        DiagReg            ;channel diagnostic register
        Filter             ;acceptance with or without a filter
        Conn_Tout          ;connection timeout: 0 for none,
                            ;otherwise, x seconds

    RES   DoSend           ;trigger for transmitting

    RES   DoDisconnect

$endinit

```

```

        COB    0
            0
read:  STH    GlobalFlags[RDATA]    ;set when a telegram arrives

        JR     L nodata

        CSF   H S.IP.Library        ;receive data
            S.IP.ReceiveData
            Channel                  ;specify number of the receive channel,
            ;on which the data
            ;arrived. For multiple channels:
            ;value of the channel register.
            RecvIP                   ;receive data from this IP
            ;address
            RecvPort                 ;receive data via this port
            4                        ;max. supported data length
            ;(buffer)
            RecvLnth                 ;effective length of received data
            RecvData                 ;received data

        SET   DoSend

nodata STH    DoSend
        ANH   DiagFlag[XCON]        ;port connected (channel)
        ANL   DiagFlag[TACT]        ;and transmitter inactive (channel)
        ANL   GlobalFlags[TBSY]     ;and transmitter free (global)
        JR     L nosend              ;sending not possible
        CSF   H S.IP.Library        ;send data
            S.IP.SendData
            Channel                  ;virtual channel
            RecvIP                   ;partner IP address
            RecvPort                 ;partner port
            4                        ;send 4 bytes
            RecvData                 ;send back the previously received
            ;4 bytes

        RES   DoSend



























        STH   DoDisconnect
        JR     L nosend

        CSF   S.IP.Library
            S.IP.DisconnectTCP
            Channel
            RecvIP
            RecvPort
        RES   DoDisconnect

nosend: ECOB

```

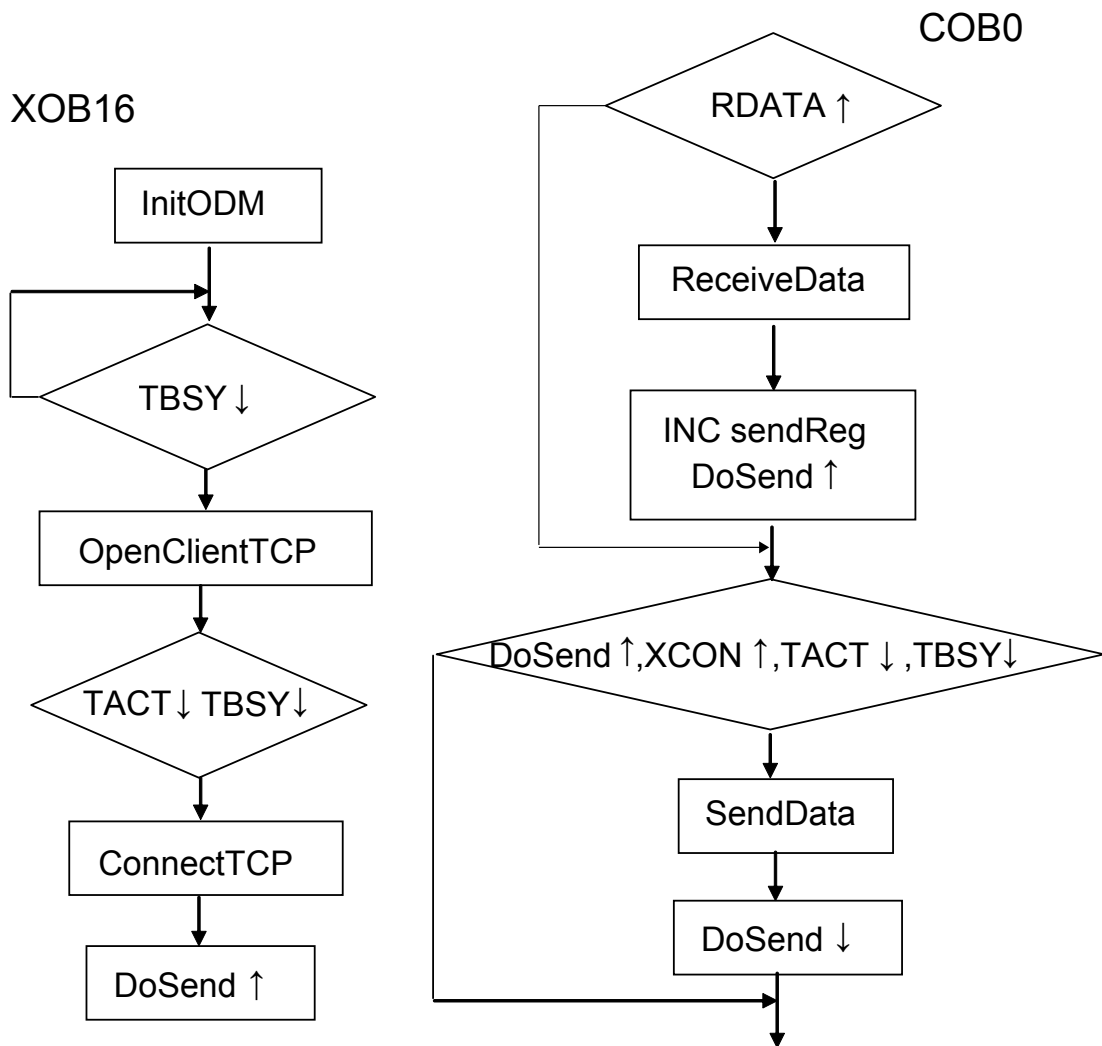
**Server code: Variable declaration and initialisation**

Group/Symbol	Type ▲	Address/Value	Comment
[-] 			
[-]  RDIA		0	
[-]  TBSY		0	
[-]  RDATA		1	
[-]  TACT		1	
[-]  RCON		2	
[-]  TDIA		2	
[-]  XCON		3	
[-]  NEXE		4	
[-]  RemotePort		65535	
[-]  GlobalFlags	F	0 [3]	
[-]  DoSend	F	5	
[-]  DiagFlag	F	10 [5]	
[-]  DoDisconnect	F	100	
[-]  Filter	K Constant	0	
[-]  Channel	K Constant	10	
[-]  Local_Port	K Constant	5001	
[-]  Conn_Tout	R		
[-]  ChannelReg	R	0	
[-]  DiagReg	R	1	
[-]  SendReg	R	2	
[-]  RecvPort	R	3	
[-]  RecvLnth	R	4	
[-]  RecvData	R	5	
[-]  RecvIP	Text RAM		
[-]  RemoteIP	Text RAM		

**6.3.2 Client**

**Client flow diagram**

- XOB16      Initialises Open Data Mode  
              Opens a TCP Client socket at port 0  
              Establishes the connection to the Server on port 5001
  
- COB0       Checks data  
              Receives data  
              Increments register (counter)  
              Sends data





**Server code listing (IL)**

“JR-1” und “JR-2” have been used for ease of presentation.

```

#include "IPLib.inc"

$init

    LD    SendReg          ;initialise the send register
        0

    LD    Conn_Tout       ;load timeout as unlimited
        0

    CSF   S.IP.Library    ;initialise Open Data Mode
        S.IP.InitODM
        GlobalFlags      ;global diag. flags
        ChannelReg       ;for multiple channels:
                        ;this is where you will find the
                        ;number of the channel in which the
                        ;received data has been written
        0                 ;timeout 0 means: remains
                        ;unlimited in mbx

    STL   GlobalFlags[TBSY] ;check if the transmitter is free
    JR    L -1

                        ;open the TCP Server port

    CSF   S.IP.Library
        S.IP.OpenClientTCP
        Channel          ;the virtual channel on which data
                        ;is sent and received
        Port             ;the local IP port over which the
                        ;data is sent and received
        DiagFlag         ;channel diagnostic flags
        DiagReg          ;channel diagnostic register
        Filter           ;acceptance with or without a filter
        Conn_Tout        ;connection timeout: 0 for none,
                        ;otherwise, x seconds

    STL   DiagFlag[TACT]
    ANL   GlobalFlags[TBSY]
    JR    L -2

    CSF   S.IP.Library    ;establish connection to the TCP Server
        S.IP.ConnectTCP
        Channel          ;virtual channel
        RemoteIP         ;Server IP address
        RemotePort       ;Server port

    SET   DoSend

    RES   DoDisconnect

$endinit

```

```

COB  0
      0
read: STH GlobalFlags[RDATA] ;set when a telegram arrives

JR   L nodata

CSF  H S.IP.Library ;receive data
      S.IP.ReceiveData
      Channel ;specify number of the receive channel,
              ;on which the data arrived
              ;arrived. For multiple channels:
              ;value of the channel register.

      RecvIP ;receive data from this IP
              ;address

      RecvPort ;receive data via this port
      4 ;max. supported data length
        ;(buffer)

      RecvLnth ;effective length of received data
      RecvData ;received data

SET  DoSend

nodata STH DoSend
      ANH DiagFlag[XCON] ;port connected (channel)
      ANL DiagFlag[TACT] ;and transmitter inactive (channel)
      ANL GlobalFlags[TBSY] ;and transmitter free (global)
      JR L nosend ;sending not possible
      CSF H S.IP.Library ;send data
            S.IP.SendData
            Channel ;virtual channel
            RecvIP ;partner IP address
            RecvPort ;partner port
            4 ;send 4 bytes
            RecvData ;send back the previously received 4
                    ;bytes

RES  DoSend


STH  DoDisconnect
JR   L nosend

CSF  S.IP.Library ;terminate connection
      S.IP.DisconnectTCP
      Channel
      RecvIP
      RecvPort
RES  DoDisconnect

nosend: ECOB






```

**Client code: Variable declaration and initialisation**

Group/Symbol	Type ▲	Address/Value	Comment
[-] 			
[-] <input type="checkbox"/> RDIA		0	
[-] <input type="checkbox"/> TBSY		0	
[-] <input type="checkbox"/> RDATA		1	
[-] <input type="checkbox"/> TACT		1	
[-] <input type="checkbox"/> RCON		2	
[-] <input type="checkbox"/> TDIA		2	
[-] <input type="checkbox"/> XCON		3	
[-] <input type="checkbox"/> NEXE		4	
[-] <input type="checkbox"/> RemotePort		5001	
[-] <input type="checkbox"/> GlobalFlags	F	0 [8]	
[-] <input type="checkbox"/> DiagFlag	F	8 [8]	
[-] <input type="checkbox"/> DoSend	F	16	
[-] <input type="checkbox"/> DoDisconnect	F	100	
[-] <input type="checkbox"/> Channel	K Constant	10	
[-] <input type="checkbox"/> Port	K Constant	5005	
[-] <input type="checkbox"/> Conn_Tout	R		
[-] <input type="checkbox"/> ChannelReg	R	0	
[-] <input type="checkbox"/> DiagReg	R	1	
[-] <input type="checkbox"/> SendReg	R	2	
[-] <input type="checkbox"/> RecvPort	R	3	
[-] <input type="checkbox"/> RecvLnth	R	4	
[-] <input type="checkbox"/> RecvData	R	5	
[-] <input type="checkbox"/> RecvIP	Text RAM		
[-] <input type="checkbox"/> RemoteIP	Text RAM		

# A Appendix

## A.1 Icons



**A.2 Contact****Saia-Burgess Controls AG**

Bahnhofstrasse 18  
3280 Murten  
Switzerland

Phone ..... +41 26 580 30 00

Fax..... +41 26 580 34 99

Email support: ..... [support@saia-pcd.com](mailto:support@saia-pcd.com)

Supportsite: ..... [www.sbc-support.com](http://www.sbc-support.com)

SBC site: ..... [www.saia-pcd.com](http://www.saia-pcd.com)

International Representatives &

SBC Sales Companies: ..... [www.saia-pcd.com/contact](http://www.saia-pcd.com/contact)

**Postal address for returns from customers of the Swiss Sales office****Saia-Burgess Controls AG**

Service Après-Vente

Bahnhofstrasse 18

3280 Murten

Switzerland

A