SIEMENS

Application Example • 02/2016

Master-Slave Communication via UDP Broadcast

SIMATIC S7-1200/S7-1500

https://support.industry.siemens.com/cs/ww/en/view/20983558

Warranty and Liability

Note

The Application Examples are not binding and do not claim to be complete with regard to configuration, equipment or any contingencies. The Application Examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for the correct operation of the described products. These Application Examples do not relieve you of the responsibility of safely and professionally using, installing, operating and servicing equipment. When using these Application Examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these Application Examples at any time and without prior notice. If there are any deviations between the recommendations provided in this Application Example and other Siemens publications – e.g. Catalogs – the contents of the other documents shall have priority.

We do not accept any liability for the information contained in this document.

Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this Application Example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act (“Produkthaftungsgesetz”), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of fundamental contractual obligations (“wesentliche Vertragspflichten”). The compensation for damages due to a breach of a fundamental contractual obligation is, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change of the burden of proof to your detriment.

Any form of duplication or distribution of these Application Examples or excerpts hereof is prohibited without the expressed consent of Siemens AG.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens’ products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit http://www.siemens.com/industrialsecurity.

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit https://support.industry.siemens.com.

Table of Contents

Warranty and Liability 2

1 Task 4

2 Solution 5

2.1 Overview 5
2.2 Description of the core functionality 6
2.3 Hardware and software components 9
2.3.1 Validity 9
2.3.2 Components used 9

3 Basics 10

3.1 Basic terms 10
3.2 UDP 11
3.3 Connections for SIMATIC controllers 11

4 Mode of Operation 14

4.1 General overview 14
4.2 User interface 17
4.3 Functionality as master 19
4.3.1 Adopting master function 19
4.3.2 Sending message frames 20
4.3.3 Slave management 23
4.3.4 Time measurement 26
4.4 Functionality as slave 27
4.4.1 Receiving of frames 27
4.4.2 Acknowledging frames 27
4.4.3 Time measurement 27
4.5 Performance characteristics 28
4.5.1 Cycle time as the master 28
4.5.2 Reaction time of the slaves 28

5 Configuration and Settings 29

5.1 Configuration of the station 29
5.2 Using the LBC library 30

6 Installation and Commissioning 34

6.1 Installing the hardware 34
6.2 Installing the software 34
6.2.1 Preparation 34
6.2.2 Loading the S7 project into the CPU 35

7 Operating the Application 37

7.1 Station as slave scenario 37
7.2 Station as master scenario 38
7.2.1 Adopting master function 38
7.2.2 Generating test data 39
7.2.3 Sending a send frame 40
7.2.4 Sending request frame 40
7.3 Diagnostics 41

8 Related literature 42

9 History 42

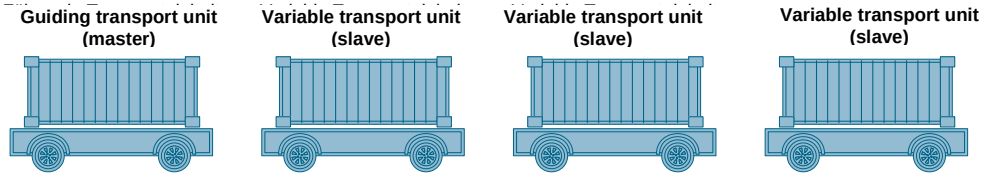
1 Task

Introduction

A frequent requirement in many applications is sending a data record quickly and simultaneously to several stations.

This could be the case in a conveyor and transport system, for example. In free-moving transport units a quasi-synchronous reaction can be initiated with only one frame. A frame could contain the information to turn all units to the left by 15°.

Figure 1-1



A combination of transport units often consists of variably combined units. Hence, it should be possible during runtime to disconnect a unit from the combination of transport units and to connect another unit.

Requirements

- Transfer of user-specific data from one master to all slaves using a broadcast
- Acknowledging the received data from the slaves to the master
- Uniform software package for master and slaves
- Efficient transfer of the data
- Adding further stations has to be possible at any time without having to modify already existing stations.
- Each station can become guiding station during operation (master-slave switchover).

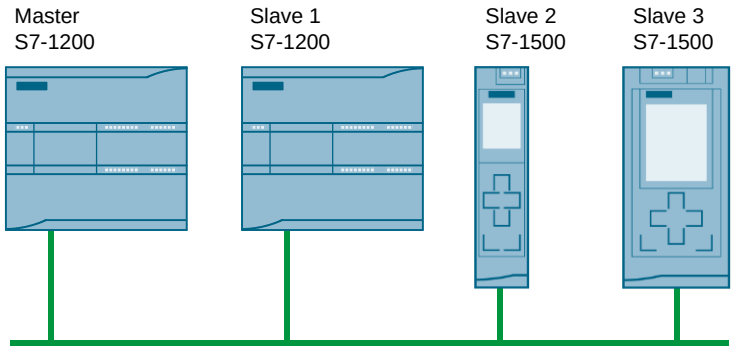
2 Solution

2.1 Overview

Setup

The application example realizes the task with SIMATIC controllers S7-1200 and S7-1500, interconnected via PROFINET:

Figure 2-1



Any number of controllers can participate here.

The UDP protocol is used for communication. This way, the connections need not be configured statically, but can be set up program-controlled (see also chapter 3.3 “Connections for SIMATIC controllers”). Hence, the participating stations need not be known to each other during the configuration process, and the master-slave switchover can be performed dynamically.

Advantages

- Already existing PROFINET/Ethernet infrastructures can be used
- No additional communication processors are required, since the PROFINET ports of the controllers are used
- Flexible topologies
- Simple expandability
- Communication bocks (TUSEND or TURCV) contained in STEP 7 are used

Assumed knowledge

The following basic knowledge is assumed:

- Handling SIMATIC controllers
- Knowledge of STEP 7 programming
- Basics of industrial communication

2.2 Description of the core functionality

Introduction

A master sends data to slaves or requests data from slaves via a UDP broadcast. The slaves acknowledge the received broadcast to the master.

The nodes need not be known during the configuration process and may change during operation. The function of the master can also be transferred to any controller during operation.

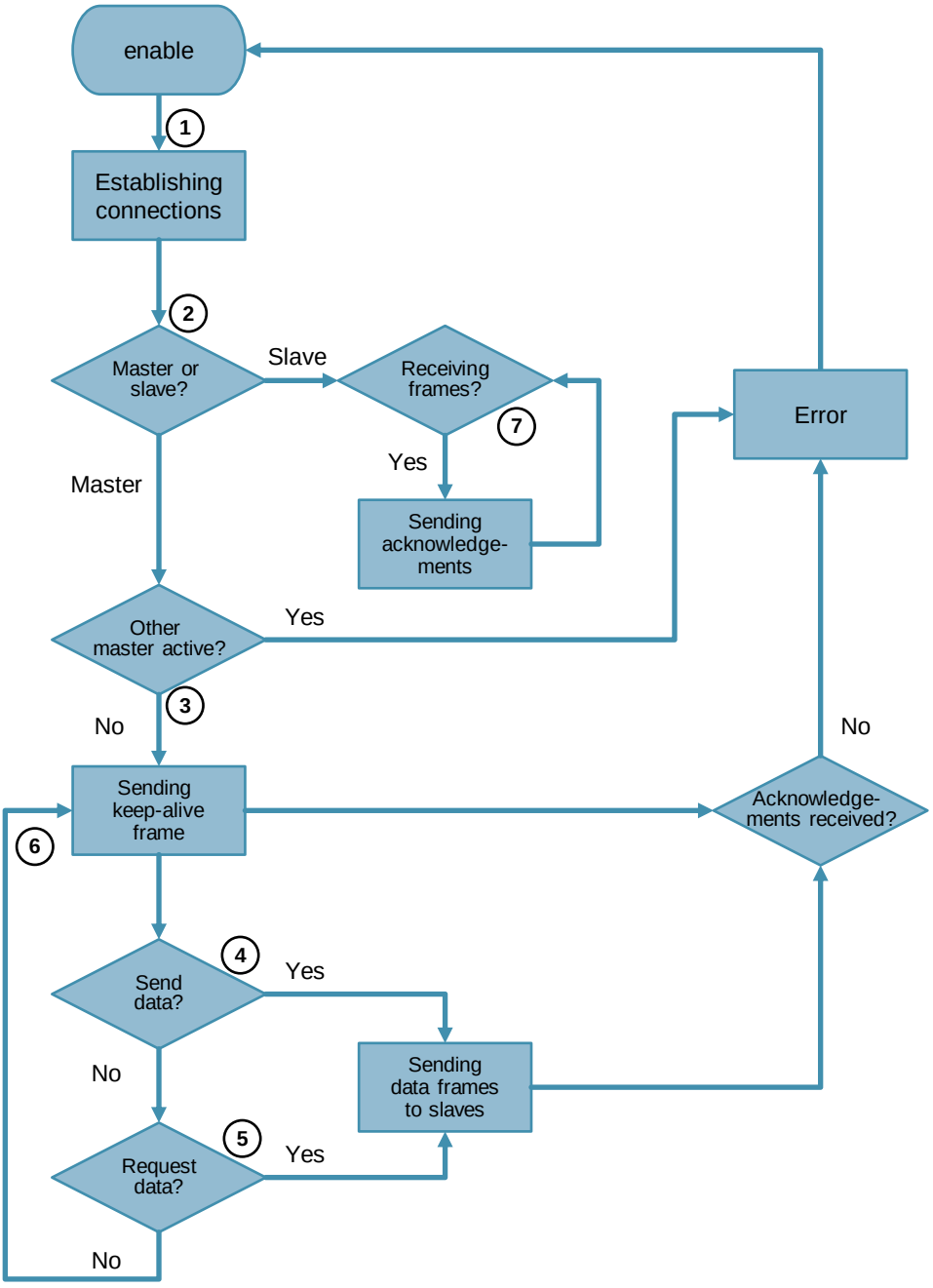
The master can send data to slaves or request data from slaves. The application sends a maximum of 236 characters per send job.

If no send jobs are pending, the master periodically polls to ensure that all nodes are available. These polls are referred to as keep-alive frames in the sections below.

Graphic program flow

This section discusses the STEP 7 program flow.

Figure 2-2



Sequence as master

Table 2-1

	Action	Note
1.	Connections are established.	
2.	If the station shall adopt the function of the master, it is checked whether another node is already active as the master.	If another master is detected in the network, an error is triggered. This does not affect the other master.
3.	If no master is active, a keep-alive frame is sent.	
4.	Data is sent to the slaves with a send command.	
5.	Data is requested by the slaves using a request command.	
6.	If send or request commands are pending, keep-alive frames are sent periodically.	

Note If not all of the known slaves confirm the receiving of a frame, an error is output. However, this does not disrupt the send operation.

Sequence as slave

Table 2-2

	Action	Note
1.	Connections are established.	
2.	If the station is not meant to take on the function as the master, it automatically takes on the function of a slave.	
7.	If the station receives a frame from the master, this will be acknowledged by a frame to the master.	

2.3 Hardware and software components

2.3.1 Validity

This application is valid for

- STEP 7 V13 SP1 or higher
- S7-1200 with firmware version 4.1 or higher, S7-1500

2.3.2 Components used

The application was created with the following components:

Hardware components

Table 2-3

Component	Qty	Article number	Note
CPU 1214C	2	6ES7214-1AG40-0XB0	FW V4.1
CPU 1511	1	6ES7511-1AK00-0AB0	
CPU 1516	1	6ES7516-1AN00-0AB0	
SCALANCE XB008	1	6GK5008-0BA00-1AB2	
Power supply 24V	1	6EP1332-4BA00	70 W 120/230 V AC

Hinweis The functionality was tested with the hardware components specified. Similar products that are not included in the above list can also be used.

Software components

Table 2-4

Component	Qty	Article number	Note
STEP 7 Professional	1	6ES7822-1AA03-0YA5	V13 SP 1

Example files and projects

The following list includes all files and projects that are used in this example.

Table 2-5

Component	Note
20983558_UDP_Broadcast_DOC_V21_en.pdf	This document
20983558_UDP_Broadcast_CODE_V21.zip	The STEP 7 project
20983558_UDP_Broadcast_LIB_V11.zip	The library created for the application example

3 Basics

3.1 Basic terms

Broadcast

If in a computer network, a data package shall be transferred from one point to all nodes in the network, then this is referred to as a broadcast. The receiver of the broadcast package needs not be specified explicitly.

Directed broadcast

The broadcast is aimed at participants of a certain network. The address for a directed broadcast to the network 192.168.0.0 with the subnet mask 255.255.255.0 is 192.168.0.255.

Port

A port is a part of a network address for assigning TCP connections, UDP connections, and data packages to server and client programs by means of operating systems. Each connection of both of these protocols includes two ports, one each on client and server side.

Socket

Together with the IP address of the station, the port number forms a so-called socket that is defined as a unique address of the user program in the overall network.
Any service of a process on a station within a network can hence be addressed with a socket.

Unicast

Unicast refers to the transmission of messages between a sender and a single receiver.

3.2 UDP

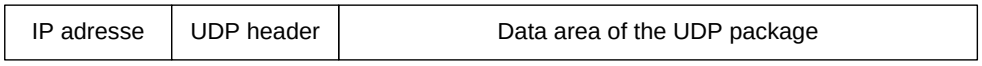
Introduction

The UDP protocol (User Datagram Protocol) was implemented for performance reasons. It offers multiple possibilities of use due to its slim frame header. In addition, almost every component in Industrial Ethernet supports the UDP protocol. The unacknowledged transfer of the data has unfavorable consequences.

Scope

Since it is required to transfer data quickly, UDP protocol only makes basic functions available. That way, data can be exchanged between communicating parties with minimum effort. UDP does not include security mechanisms as available in TCP/IP.

Figure 3-1



In addition, the UDP protocol is connectionless and non-stream-oriented, i.e. data packets are sent as a whole.

Disadvantages

Due to the lack of security mechanisms, the resulting disadvantages have to be taken into consideration during usage and may require a respective practical solution.

- There is no renewed sending of lost data packets.
- Data packets with incorrect checksum are rejected and not newly requested.
- Multiple deliveries of individual packets are possible due to the properties of the IP protocol as subordinate protocol.
- The arrival sequence of the data at the receiver cannot be predicted.

3.3 Connections for SIMATIC controllers

Introduction

A connection defines a logic assignment of two communication partners for executing communication services. A connection defines the following:

- Communication partners involved
- Type of connection (e.g. S7 connection)
- Special properties (e.g. whether a connection remains permanent or is established and terminated dynamically in the user program, and whether operating status messages shall be sent)
- Connection path

Configuration of connections

Communication connections can be configured or, under certain circumstances, also be generated program-controlled.

The addresses of the local and the remote connection partner are given when networking the devices and assigned to the connections during configuration.

Exceptions here are the following connections:

- Free UDP connection: For the free UDP connection, the address is specified at the communication interface in the user program.
- Programmed communication connection

Fully specified connections

Fully specified connections have the following properties:

- The addresses and the network parameters of the local and the remote communication partner are defined.
- After loading the connection parameters, the communication connection is ready for operation.

Unspecified (partly specified) connections

Unspecified connections have the following properties:

- Only the local communication partner is defined.

  This may have the following reasons:

  - the partner for the selected connection type is not networked
  - the partner is located outside the project
  - the partner is not meant to be defined due to a specific handling procedure
- After loading the connection parameters, the communication connection is only partly ready for operation.

Application cases:

- The hardware configuration is still incomplete, so complete networking of the devices is not yet possible.
- Ready to receive mode shall be established for any (unspecified) communication partner.

Programmed communication connection

Programmed communication connections are unspecified or partly specified connections.

In certain application areas, it is an advantage not to establish the communication connections statically via the configuration. Therefore, as an alternative, there is the option of setting up certain connection types program-controlled via a specific application, hence, dynamically on demand.

Instructions

The table lists the STEP 7 instructions for using open communication services (Open User Communication) with UDP.

Table 3-1

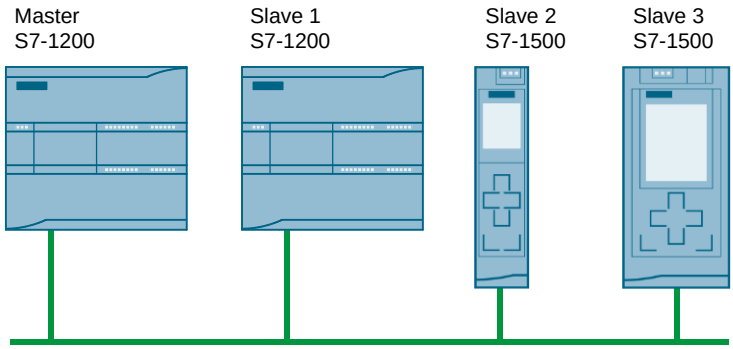
Designation	Short description
TCON	Connection establishment
TDISCON	Disconnecting the process
TUSEND	Sending data; connectionless UDP protocol in compliance with RFC 768
TURCV	Receiving data; connectionless UDP protocol in compliance with RFC 768

4 Mode of Operation

4.1 General overview

Introduction

Figure 4-1



Several controllers are interconnected via PROFINET. Each controller can temporarily take on the function of the master, whereas only one master at a time must be active.

The master sends data to the slaves or requests data from slaves via a UDP broadcast. The slaves acknowledge the received broadcast with a message frame to the master.

The data of the slaves is managed by the master and stored in a data block.

Communication

In this application example, the following programmed communication connections are used. This has the advantage that the respective other controllers need not be known during the configuration process, and any further controllers can be supplemented.

For sending and receiving frames, the communication blocks TUSEND and TURCV that are integrated in STEP 7 are used here.

Note This application example realizes a directed broadcast. Thus the communication is limited to a subnet. All participants must use the same subnet.

Data

Per send job, 236 characters of user data are transferred between master and slave. The number of characters to be transferred can be adjusted in data type “LBC_typeUserData”. The size of the array is calculated by the respective function blocks during initialization, and the parameters of the TUSEND and TURCV blocks are set accordingly.

More information on the message frame structure is available in the chapters 4.3.2 “Sending message frames” and 4.4.2 “Acknowledging frames”.

Program overview

For this application example, the “LBC” block library was created: Blocks that are part of the library are labeled with prefix “LBC”. The library is not know-how protected and can be downloaded separately (see \2\).

Figure 4-2

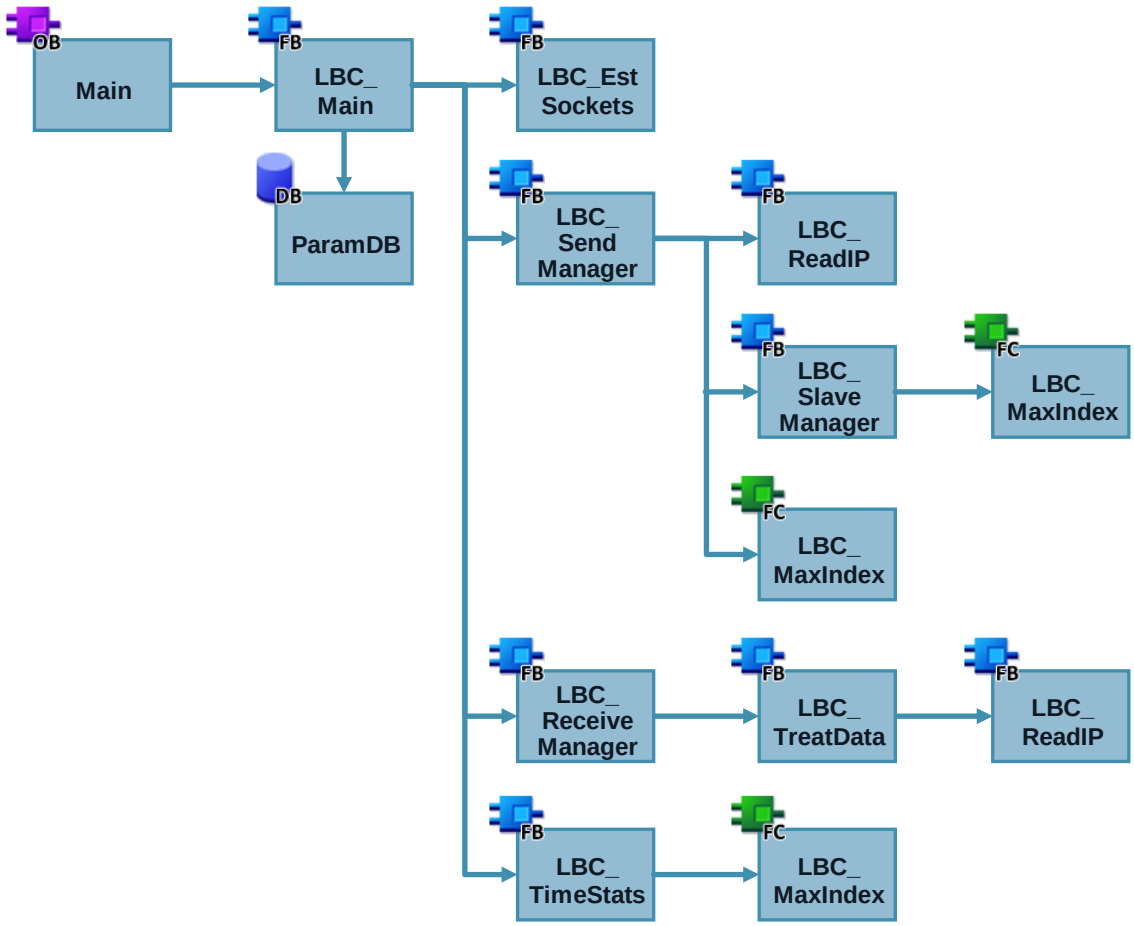


Table 4-1

Symbolic name	Function
LBC_Main	User interface, central block: • Evaluating the user parameters • Defining the master/slave status • Calling the subordinate function blocks • Coordination of keep-alive frames, send requests, or requests • Output of user parameters
LBC_EstSockets	Establishes two connections for sending and receiving frames. The connections are then used as follows: Connection 1: • As master: sending a broadcast to all nodes in the subnet

Symbolic name	Function
	• As slave: receiving a broadcast frame Connection 2: • As master: receiving the acknowledgement of the slaves • As slave: sending the acknowledgement to the master
LBC_SendManager	Realizing the sending of frames as the master: • Creating the user data of the broadcast frame • Sending the broadcast frame • Calling LBC_SlaveManager
LBC_ReadIP	Reads the IP address of the internal PROFINET interface from the S7-CPU. The own IP address is written to the broadcast frame, or to the acknowledgement of the slaves respectively.
LBC_SlaveManager	Manages the slaves array: • Receiving the acknowledgement of the slaves • Assignment of incoming frames to the elements of the slaves array
LBC_ReceiveManager	Realizes the slave functions and checks, whether another master exists in the network: • Receives broadcast frames from the master • Calls LBC_TreatData • Sends a response/acknowledgement frame to the master • Reports if a frame from a master has arrived
LBC_TreatData	Evaluates the received frames from the master and generates the acknowledgement.
LBC_TimeStats	Performs time measurements: • As master: time between sending a frame and receiving all acknowledgements of the active slaves • As slave: time between receiving two frames from a master
LBC_MaxIndex	Calculates the number of elements of an array

4.2 User interface

Function block “LBC_Main” is the central block of the program and us used as user interface. With this block, the parameters for operating the parameter are transferred, and any of the further contained blocks are called.

Interface

Figure 4-3

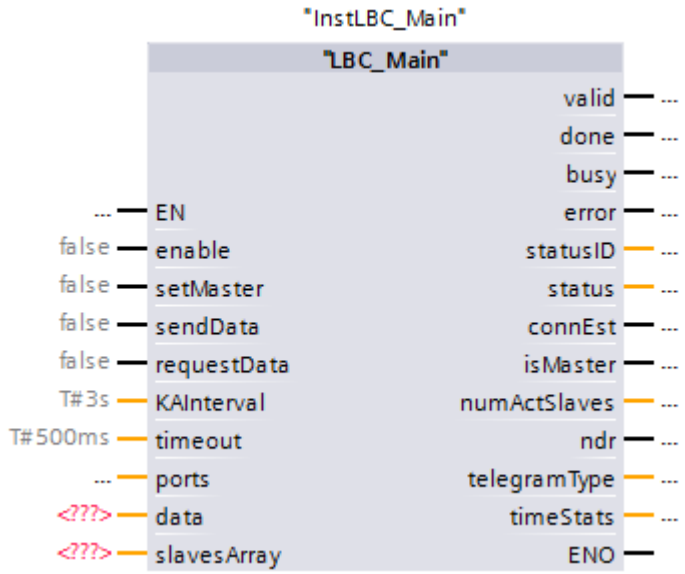


Table 4-2

Symbol	Type	Notes
enable	Input Bool	Activates the program. As long as #enable is active, the master periodically sends keep-alive frames, and the slaves acknowledge these. A falling edge at #enable resets the blocks as well as the following parameters: • Time measurement • Counters of the received frames • Content of the slave array (slave array reinitialized with the next frame)
setMaster	Input Bool	For #setMaster = TRUE, the device becomes the master if no other master is active.
sendData	Input Bool	A positive edge at #sendData sends data to the slaves. In slave mode, the input is irrelevant.
requestData	Input Bool	A positive edge at #requestData requests the data of the slaves. In slave mode, the input is irrelevant.

Symbol	Type	Notes
KAInterval	Input Time	Specifies the interval in which the keep-alive frames are sent to the slaves. These parameters need to be identical for all nodes to ensure that an active master is detected in time, before a controller can become the master. In slave mode, the input is irrelevant. Min.-value: 200 ms.
timeout	Input Time	Specifies after which time all acknowledgements from the slaves need to have arrived before an error is output. In slave mode, the input is irrelevant. Min.-value: 100 ms.
ports	Input “LBC_typePorts”	Specifies which ports shall be used for the communication between master and slaves.
valid	Output Bool	Indicates, that the parameters are permitted and the program can be executed.
done	Output Bool	As master: broadcast frame was sent successfully. The bit is only set for one cycle.
busy	Output Bool	The program is busy.
error	Output Bool	An error was detected. The bit is active for as long as #enable remains set.
statusID	Output UInt	Specifies the source of the status messages. The output is active for as long as #enable remains set.
status	Output Word	Outputs status messages and in an #error event, it specifies the error. The output is active for as long as #enable remains set.
connEst	Output Bool	The connections have been established.
isMaster	Output Bool	#isMaster = TRUE; the device works as master. #isMaster = FALSE; the device works as slave.
numActSlaves	Output Int	Specifies the number of active slaves.
ndr	Output Bool	New frame received from the master. The bit is only set for one cycle. In master mode, the output is irrelevant.
telegramType	Output Int	Displays the type of the received frame for #ndr = TRUE. 1: sendData 2: requestData 3: KA frame In master mode, the output is irrelevant.
timeStats	Output “LBC_typeTimeStats”	Outputs time measurements.

Symbol	Type	Notes
data	InOut “LBC_ typeUserData”	Master: data to be sent. Slave: data received.
slavesArray	InOut “LBC_typeSlavesArray”	Array with information and user data of the slaves.

4.3 Functionality as master

4.3.1 Adopting master function

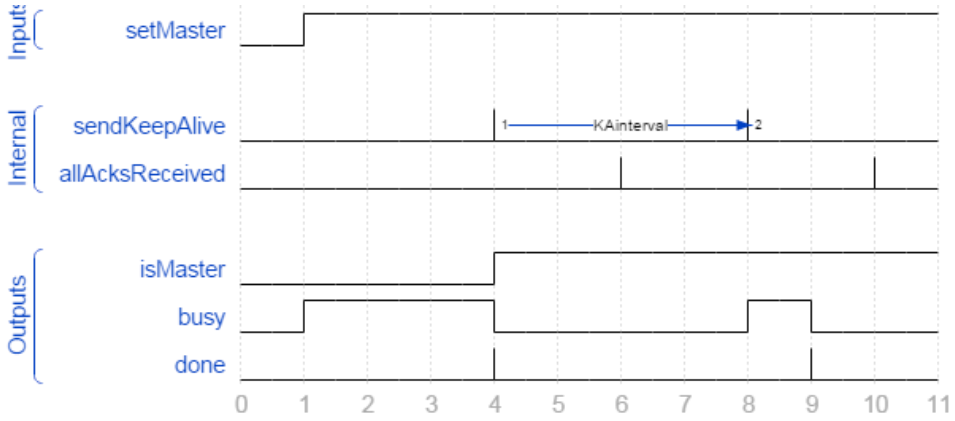
Any controller can become the master temporarily. However, only one master at a time can be active.

If input #enable has been set, the application is started and the station establishes the required connection. By setting the #setMaster input, it is checked whether a master is already active in the network. The configured time #KAInterval is waited. If within this time no frame is received by a master, then there is no active master and the controller takes on the function of the master. The fact that the controller has taken on the function of the master is indicated at output #isMaster.

NOTICE	Parameter #KAInterval must be identical for all nodes to guarantee, that an active master is detected in time, before a controller can become the master.

The following graphic shows how the master function is taken on and the keep-alive frames are sent.

Figure 4-4



4.3.2 Sending message frames

General

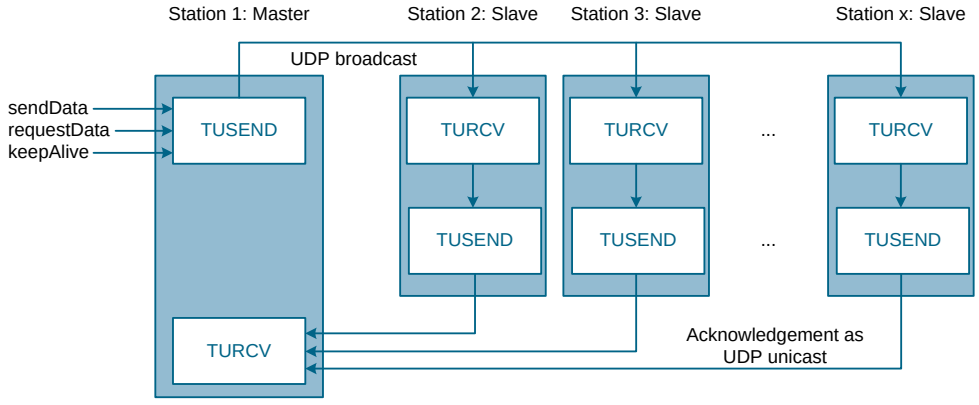
With the user interface of the master, the user can send the following frames:

- sendData: sending data to the slaves
- requestData: requesting data from the slaves

If no job is pending, keep-alive frames are automatically sent in configured time intervals in order to poll the status of the slaves.

Any frame from the master is always acknowledged by the slaves. The time of receiving the acknowledgements is monitored by the master.

Figure 4-5



If several send jobs are active at the same time, the program prioritizes the send jobs as follows:

1. Send job
2. Request job
3. Keep-alive frame

As soon as a frame was sent successfully, the block is ready to send another frame. This is signaled by output #done which is active for one respective cycle (see Figure 4-4). The receiving of the acknowledgements is not waited for here.

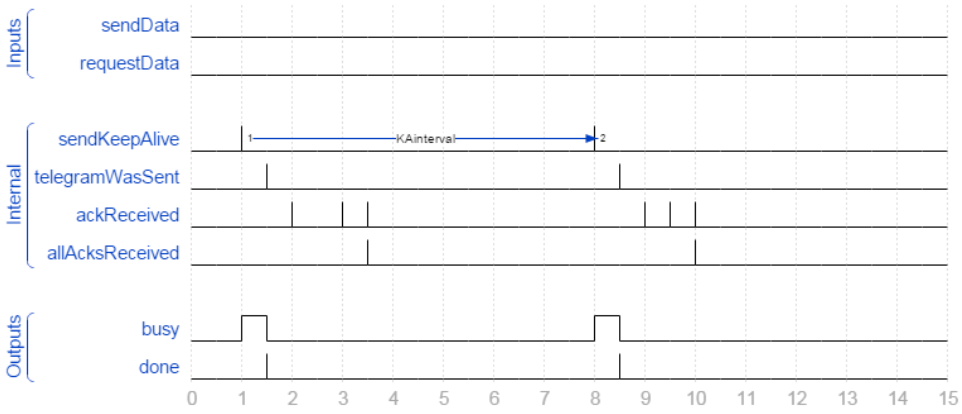
Keep-alive

Keep-alive frames are periodically sent from the master and the reception acknowledged by the slaves. This enables the master to ensure, even during a send pause, that all slaves are still active; otherwise, it can also detect on short notice, whether the connection to a slave was disrupted.

The interval for sending keep-alive frames is defined via the #KAInterval input.

These frames are only sent if currently no send or request frames shall be sent.

Figure 4-6



Send

A sent frame transfers data from the master to the slaves. This is triggered with a positive edge at the #sendData input.

The user data to be transferred is supplied to InOut parameter #data as data type “LBC_typeUserData”. This data type consists of an array of 236 character (Chars), that can be transferred.

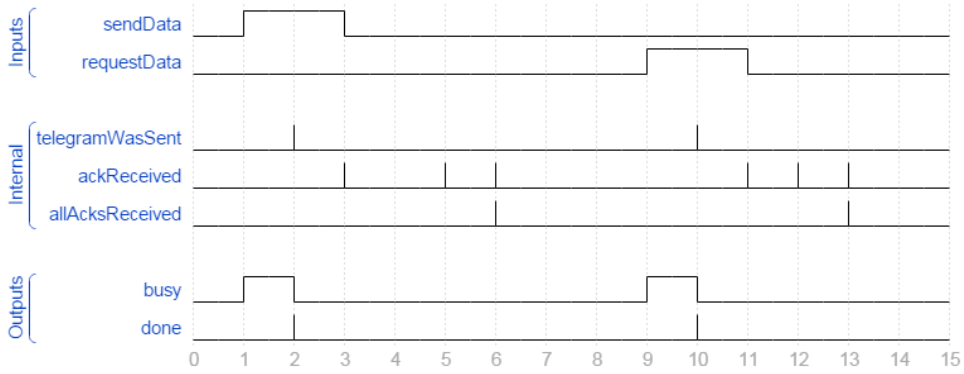
In addition to the user data, the following data is transferred in a frame:

- Source address of the master (so the slaves know which address to send the acknowledgement to)
- Time stamp
- Frame type (keep-alive, send or request)

Figure 4-7

Source address	Time stamp	Frame type	User data

Figure 4-8



Request

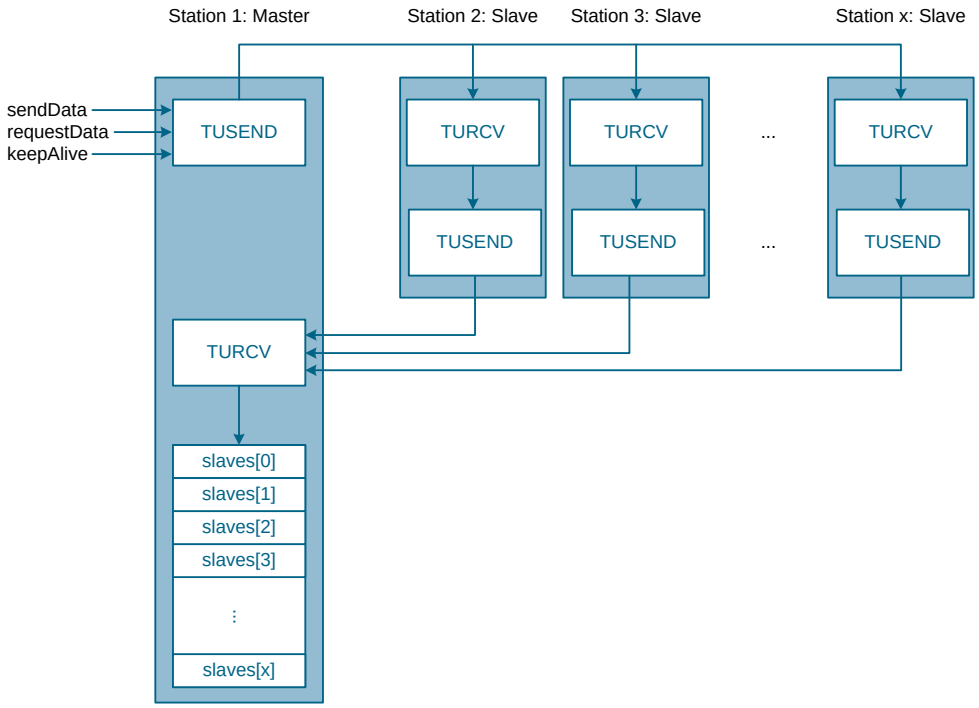
The master uses a request frame to request data from the slaves. The received data is stored in the slave management of the master.

Sending a request frame is triggered with a positive edge at the #requestData input (see Figure 4-8).

4.3.3 Slave management

Due to the received acknowledgements, the master knows all the slaves and stores them in an array. This is performed in function block “LBC_SlaveManager”.

Figure 4-9

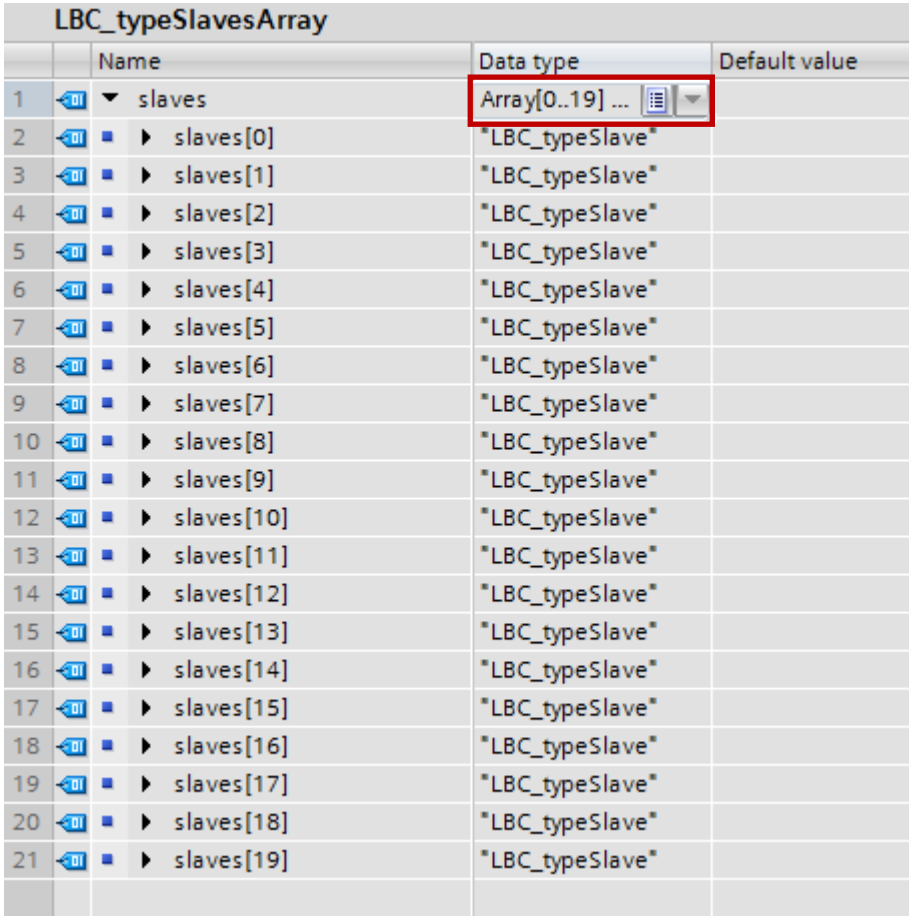


This slave array is defined by data type “LBC_SlavesArray” and receives the following data:

- IP address and port of the slave
- Status, whether the slave is active
- Status, whether the slave has acknowledged the receiving of the last frame
- Data received from the slave
- Number of received keep-alive frames of the slave
- Number of received request and send frames of the slave

Since the memory must not be allocated during the runtime of the controller, the maximum number of slaves to be managed must be known during the configuration process. This can be adjusted in data type “LBC_typeSlavesArray”. In the delivery state of the application example, a maximum number of 20 slaves has been configured.

Figure 4-10



LBC_typeSlavesArray

	Name	Data type	Default value
1	slaves	Array[0..19] ...	
2	slaves[0]	"LBC_typeSlave"	
3	slaves[1]	"LBC_typeSlave"	
4	slaves[2]	"LBC_typeSlave"	
5	slaves[3]	"LBC_typeSlave"	
6	slaves[4]	"LBC_typeSlave"	
7	slaves[5]	"LBC_typeSlave"	
8	slaves[6]	"LBC_typeSlave"	
9	slaves[7]	"LBC_typeSlave"	
10	slaves[8]	"LBC_typeSlave"	
11	slaves[9]	"LBC_typeSlave"	
12	slaves[10]	"LBC_typeSlave"	
13	slaves[11]	"LBC_typeSlave"	
14	slaves[12]	"LBC_typeSlave"	
15	slaves[13]	"LBC_typeSlave"	
16	slaves[14]	"LBC_typeSlave"	
17	slaves[15]	"LBC_typeSlave"	
18	slaves[16]	"LBC_typeSlave"	
19	slaves[17]	"LBC_typeSlave"	
20	slaves[18]	"LBC_typeSlave"	
21	slaves[19]	"LBC_typeSlave"	

Initialization

In order to make the number of slaves adjustable during runtime, a dynamic structure of the slave array must be established. In an initialization process, the slaves are addressed via a keep-alive frame, and the slave array is filled up in the order of the incoming responses from the slaves.

Since the response order may vary, the slave array may be structured differently for each initialization process.

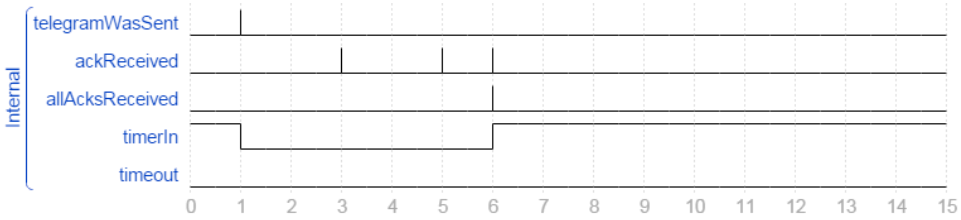
After initializing the slave array, the responses from the slaves are compared with the known IP addresses for each frame and hence assigned to the respective slave.

Modules of a slave

The acknowledgements from the slaves are time monitored. The successful sending of a frame starts a timer with the configured time #timeout. If, within this time, all acknowledgements from the active slaves arrive, the timer is reset. When sending a frame next time, the timer is started again.

The following graphics shows the receiving of all acknowledgements for three active slaves within the configured time.

Figure 4-11

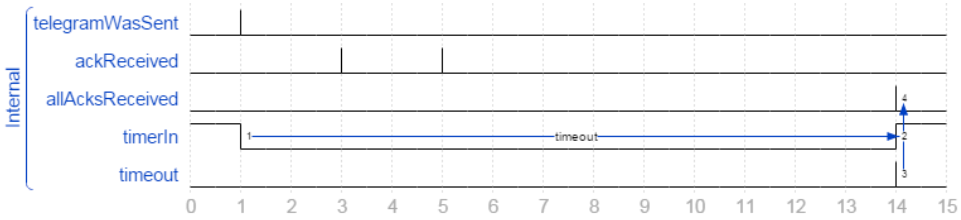


If a slave does not respond within the configured response time #timeout, the following happens:

- A status message is output at output #status (see chapter 7.3).
- The #isActive bit is reset in the slave array of the respective slave.
- The number of active slaves #numActSlaves is reduced by 1.
- The element in the slave array remains.

The following graphic shows the receiving of only two acknowledgements for three active slaves and the elapsed configured time #timeout.

Figure 4-12



Return of a slave

If an inactive slave responds to a frame again, the following happens:

- A status message is output at output #status (see chapter 7.3).
- The #isActive bit is set again in the slave array of the respective slave.
- The number of active slaves #numActSlaves is incremented by 1.
- The previously used element is further used in the slaves array.

Response of an unknown slave

If during operation, a previously unknown slave answers, the following happens:

- A status message is output at output #status (see chapter 7.3).
- The information of the slave is stored in a new element of the slave array.
- The #isActive bit is set in the slave array of the respective slave.
- The number of active slaves #numActSlaves is incremented by 1.

4.3.4 Time measurement

The program as the master measures the time between sending a frame and receiving all acknowledgements from the known slaves. The result of the measurement is output at output #timeStats in the format “LBC_typeTimeStats”.

This contains the current measured value, the last 20 measured values, and the averaged measured value over the last 20 measured values.

The number of measured values to be stored can be adjusted in data type “LBC_typeTimeStats”.

4.4 Functionality as slave

If input #enable has been set, the application is started and the station establishes the required connection. If the station shall not be a master and therefore, #setMaster is not set, the station takes on the function of a slave and responds to all frames arriving at the configured port.

4.4.1 Receiving of frames

A station in slave mode cyclically calls function block “LBC_ReceiveManager” in which frames are received and acknowledged. At output #ndr, the slave indicates for one cycle that a new frame was received. Output #telegramType specifies the type of the frame.

The received user data is provided at InOut parameter #data.

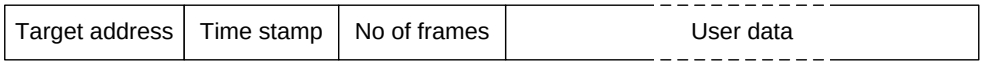
4.4.2 Acknowledging frames

All frames are always acknowledged by the slaves. The size of the acknowledgement is always the same, irrespective of the type of the frame that is acknowledged.

Acknowledgement consists of the following data:

- Target address of the master
- Time stamp
- Number of received frames
- User data to be transferred (the user data is only updated with the response to request frames)

Figure 4-13



4.4.3 Time measurement

The program as the slave measures the time between two received frames from the master. The result of the measurement is output at output #timeStats in the format “LBC_typeTimeStats”.

This contains the current measured value, the last 20 measured values, and the averaged measured value over the last 20 measured values.

4.5 Performance characteristics

4.5.1 Cycle time as the master

CPU 1214C

Table 4-3

Type	Cycle time in ms
shortest	2
average	3
longest	8

CPU 1516

Table 4-4

Type	Cycle time in ms
shortest	0
average	1
longest	3

4.5.2 Reaction time of the slaves

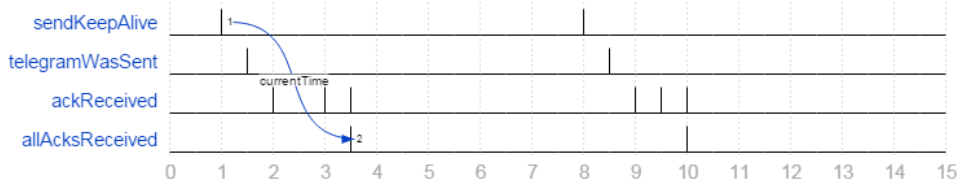
The time between sending periodical keep-alive frames by the master, and receiving all acknowledgements from the slaves with three slaves active is measured.

Table 4-5

Type	Reaction time in ms
shortest	19
average	44
longest	84

Note The mean value of the reaction times was determined from 100 measured values, while every 500 ms a keep-alive frame was sent.

Figure 4-14



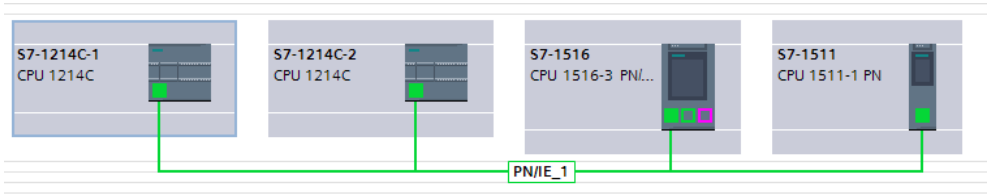
5 Configuration and Settings

The enclosed project does not require any further configuration. If you want to replicate the application example with other components, then the most important settings are shown in this chapter.

5.1 Configuration of the station

1. Open TIA Portal and create a new project.
2. Open “Devices & network” in the project tree.
3. Drag the controllers used from the hardware catalog into the workspace.
4. Alternatively, you can also create a separate project for each controller. Please ensure, that each controller has a unique IP address and the controllers are located in the same subnet.
5. Connect the PROFINET interfaces of the created controllers with each other if your project contains several controllers.
   This step is mandatory. However, different IP addresses are then automatically assigned to the controllers.

Figure 5-1



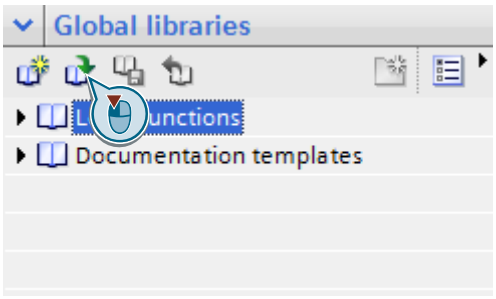
Note If further controllers should be added to the configuration, the configuration of the already existing configuration needs not be adapted. Also, each controller can be configured in a separate project without knowing the remaining controllers.

5.2 Using the LBC library

Downloading the library

1. Download the LBC library and unzip the file. The download can be found under \2\.
2. In TIA Portal V13, you open the “Libraries” area in the right window pane.
3. Expand the “Global libraries” tab.
4. Click on the “Open global library” icon and select the extracted library:

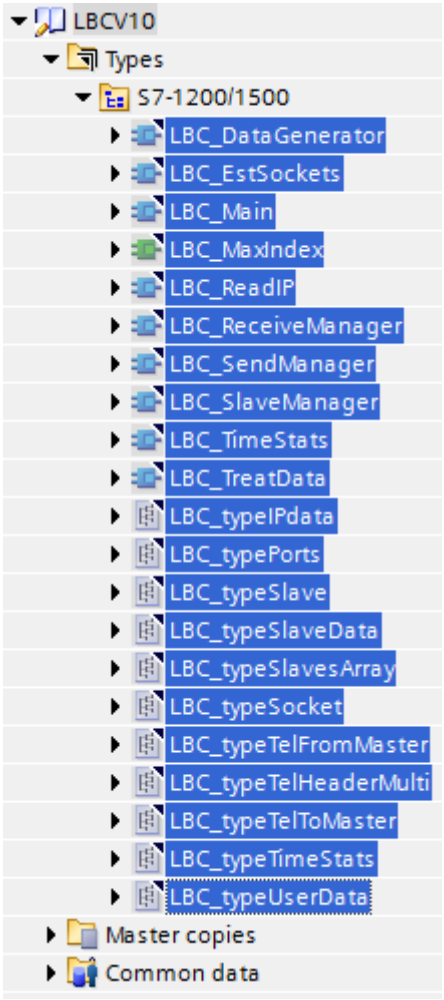
Figure 5-2



Instantiating the library blocks

1. Expand the folder of one of your controllers in the project tree.
2. Unfold the “LBC” library and the “Types” and “S7-1200_S7-1500” folder, and drag its content to the “Program blocks” folder of the respective controller. The program blocks and data types of the library are now instanced.

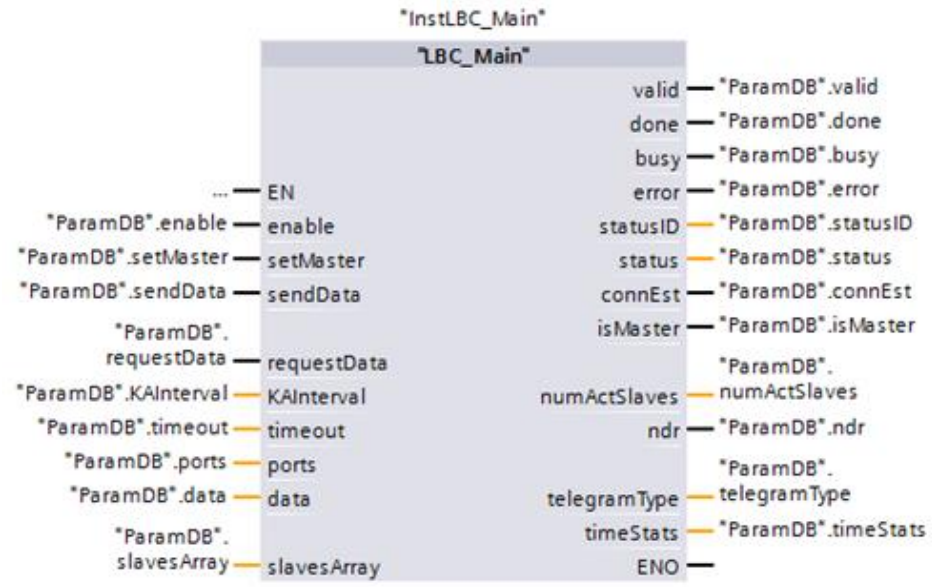
Figure 5-3



3. In “Program blocks” you open organization block OB1.
4. From folder “Program blocks” you drag function block “LBC_Main” into a free network and create an instance data block.
5. Assign actual parameters to the formal parameters of the function block.

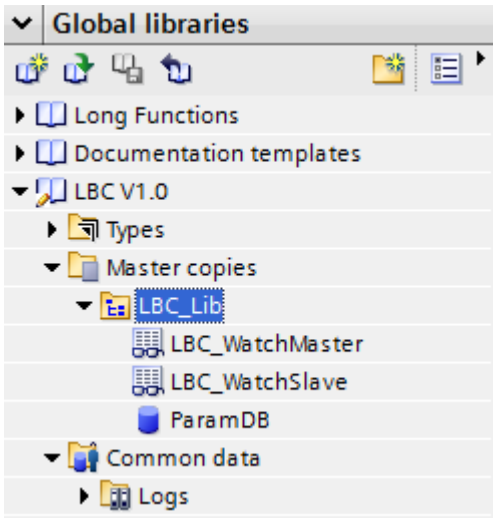
   For test purposes, you can use the actual parameters of data block “LBC_ParamDB” from the “Master copies” folder of the library. Drag them from the library onto the “Program blocks” folder.

Abbildung 5-4



6. Drag watch table “LBC_WatchMaster” and “LBC_WatchSlave” from the “Master copies” folder of the library onto the “Watch and force tables” folder.

Figure 5-5



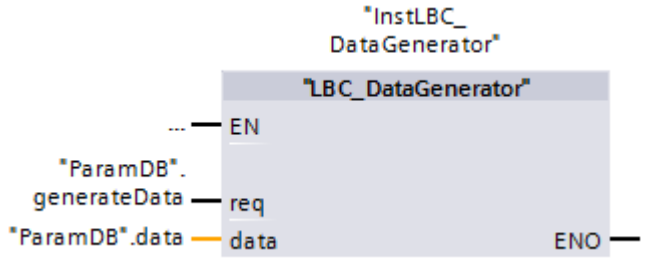
7. Repeat steps 5 to 9 for the other controllers in your project.

Generating test data

Execute the following steps, if the application shall generate some test data for test purposes.

1. Open OB1 or create another organization block in whose cycle some test data shall be generated.
2. Drag function block “LBC_DataGenerator” from folder “LBC_Lib” in “Program blocks” into an empty network of the organization block.
3. Assign actual parameters of data block „LBC_ParamDB” to the formal parameters of the function block.

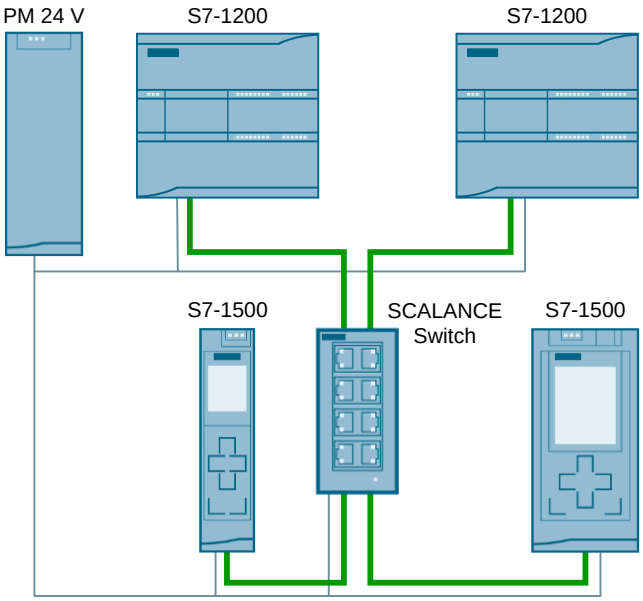
Figure 5-6



6 Installation and Commissioning

6.1 Installing the hardware

Figure 6-1



1. Mount the controllers, power supply unit, and switch onto DIN rails.
2. Connect the 24V DC supply voltage to controllers and switch.
3. Connect the PROFINET ports of the controllers with the SCALANCE switch.

6.2 Installing the software

6.2.1 Preparation

1. Download the file 20983558_MasterSlave_UDP-Broadcast_CODE_V21.zip. The download can be found under \2\.
2. Save the zip files to any directory on your computer and extract them.
3. Set the IP address of the PG/PC so the PG/PC is located in the same subnet as the CPUs.
4. Use an Ethernet cable to connect the PG/PC with the SCALANCE switch.

For this application example, the following IP addresses were used:

CPU 1214C (1)
IP address: 192.168.0.10
Subnet mask: 255.255.255.0

CPU 1214C (2)
IP address: 192.168.0.11
Subnet mask: 255.255.255.0

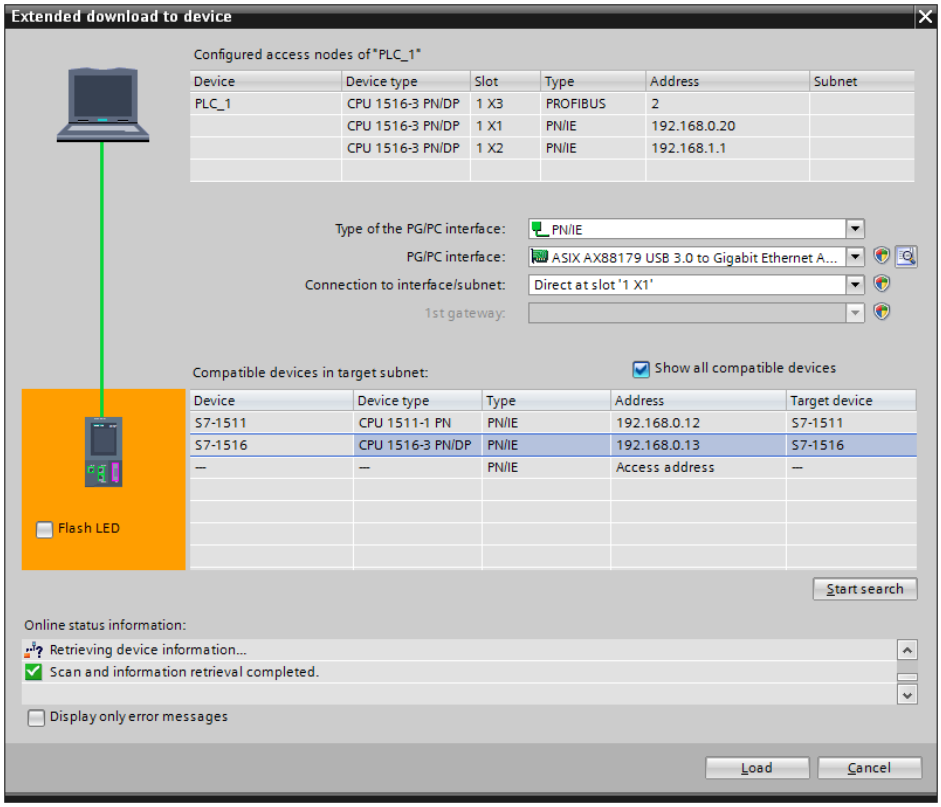
CPU 1511
IP address: 192.168.0.12
Subnet mask: 255.255.255.0

CPU 1516
IP address: 192.168.0.13
Subnet mask: 255.255.255.0

6.2.2 Loading the S7 project into the CPU

1. Open TIA Portal.
2. Go to the project view.
3. Click “Project > Open” in the menu bar in the TIA Portal.
4. Click “Browse” and open the respective project.
5. Set the respective CPU to STOP.
6. Right click on the respective CPU in the project tree and then on “Download to device > Hardware and Software (only changes)”.
7. Select the respective interface and click “Start search”.

Figure 6-2



8. Select the CPU based on the address and then click on “Load”.

Note The IP address and the device name are automatically assigned when downloading the project into the CPU.

9. Confirm the dialog by clicking “Load”.
10. Click “Finish” when the loading process is complete.

7 Operating the Application

Operating the application for test purposes via watch tables is discussed below. These are already contained as copy templates in the supplied library (see \2\).

Copy the watch tables “LBC_WatchMaster” and “LBC_WatchSlave” for any participating controller in the respective project folders (see chapter 5.2).

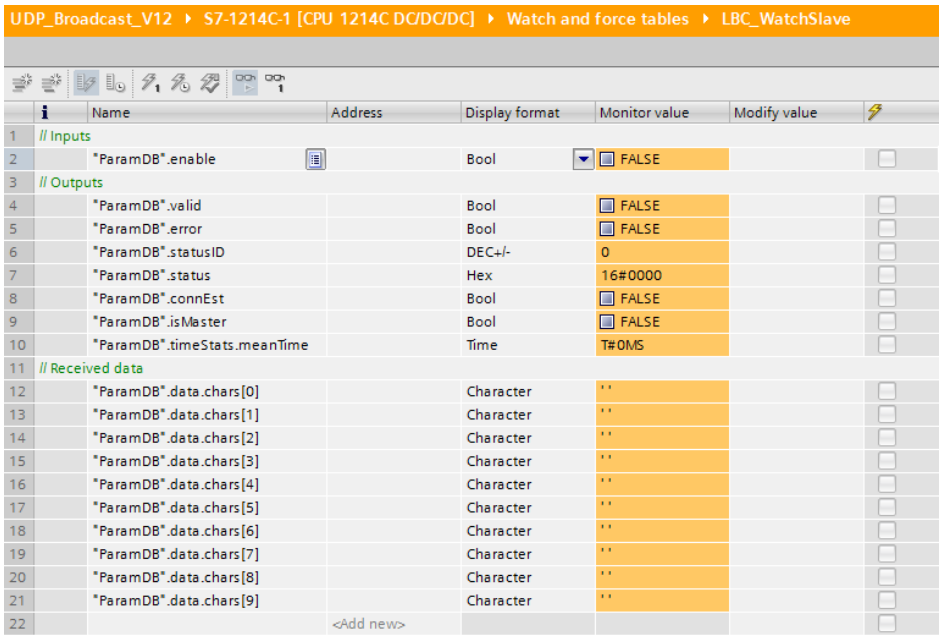
7.1 Station as slave scenario

This chapter explains how the slave function is activated for a station.

Table 7-1

No.	Action	Notes
1.	Open the watch table “LBC_WatchSlave” of the respective controller.	The watch table is contained in the copy templates of the supplied library (see chapter 5.2).
2.	Control the „LBC_ParamDB”.enable tag to TRUE.	This starts the application and the connection is established. As soon as the connections are established, the station receives frames and acknowledges them. The received data is represented in array „LBC_ParamDB”. The average time between receiving two frames from a master is displayed in the „LBC_ParamDB”.timeStats.meanTime tag.

Figure 7-1



	Name	Address	Display format	Monitor value	Modify value
1	// Inputs				
2	"ParamDB".enable		Bool	FALSE	
3	// Outputs				
4	"ParamDB".valid		Bool	FALSE	
5	"ParamDB".error		Bool	FALSE	
6	"ParamDB".statusID		DEC+/-	0	
7	"ParamDB".status		Hex	16#0000	
8	"ParamDB".connEst		Bool	FALSE	
9	"ParamDB".isMaster		Bool	FALSE	
10	"ParamDB".timeStats.meanTime		Time	T#0MS	
11	// Received data				
12	"ParamDB".data.chars[0]		Character	''	
13	"ParamDB".data.chars[1]		Character	''	
14	"ParamDB".data.chars[2]		Character	''	
15	"ParamDB".data.chars[3]		Character	''	
16	"ParamDB".data.chars[4]		Character	''	
17	"ParamDB".data.chars[5]		Character	''	
18	"ParamDB".data.chars[6]		Character	''	
19	"ParamDB".data.chars[7]		Character	''	
20	"ParamDB".data.chars[8]		Character	''	
21	"ParamDB".data.chars[9]		Character	''	
22		<Add new>			

7.2 Station as master scenario

7.2.1 Adopting master function

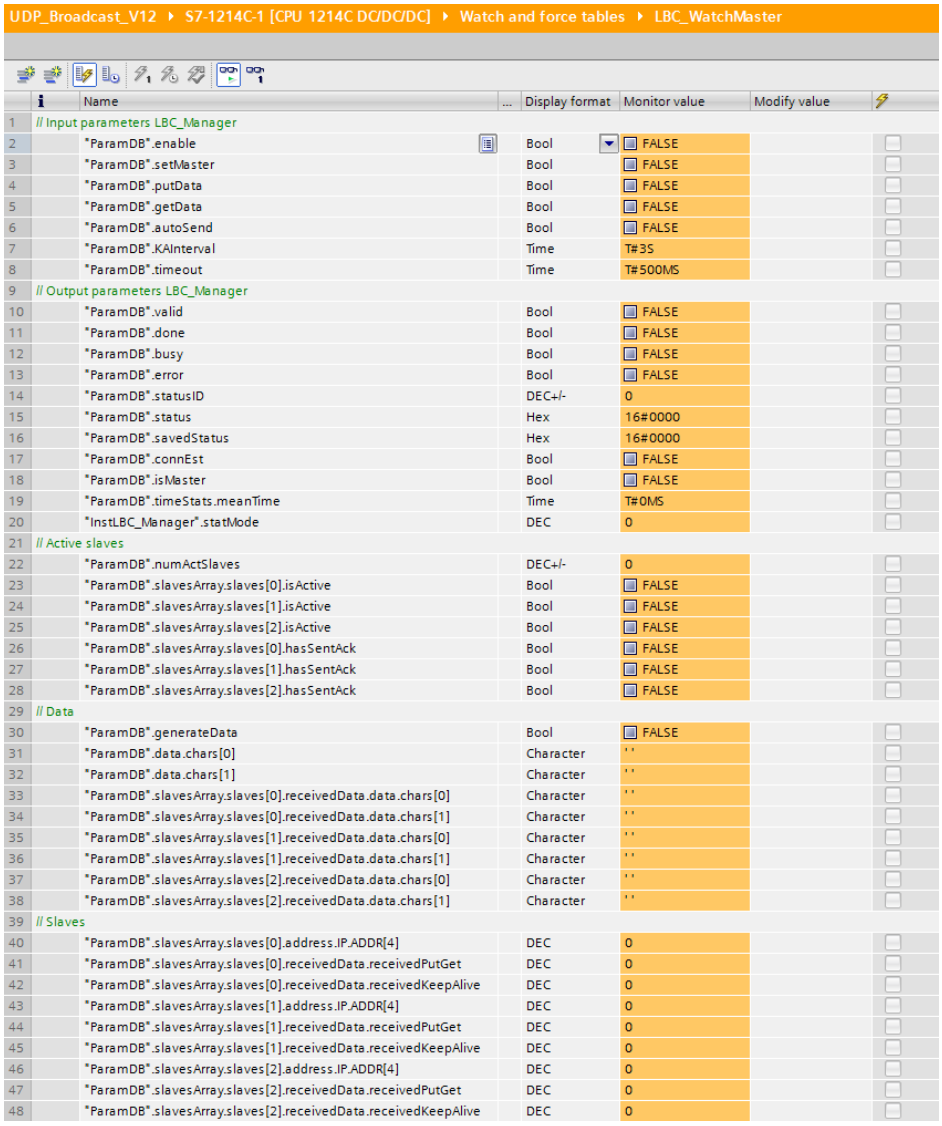
This chapter explains how the master function is activated for a station.

Table 7-2

No.	Action	Notes
1.	Open watch table “LBC_WatchMaster” of the respective controller.	The watch table is contained in the copy templates of the supplied library (see chapter 5.2).
2.	Control the „LBC_ParamDB”.enable tag to TRUE.	This starts the application and the connection is established.
3.	Control the „LBC_ParamDB”.setMaster tag to TRUE.	The station now waits for the configured time #KAinterval to check whether a master is already active. If within this time no frame is received by a master, then there is no active master, and the controller takes on the function of the master and the „LBC_ParamDB”.isMaster tag is set. From now on, keep-alive frames are periodically sent by the master. With the receiving of the acknowledgements, the #isActive tags of the slave array are set, and the number of active slaves are displayed in „LBC_ParamDB”.numActSlaves. The average time between sending a

No.	Action	Notes
		frame and receiving all acknowledgements from the active slaves is displayed in the „LBC_ParamDB”.timeStats.meanTime tag.

Figure 7-2



	Name	Display format	Monitor value	Modify value
1	// Input parameters LBC_Manager			
2	"ParamDB".enable	Bool	FALSE	
3	"ParamDB".setMaster	Bool	FALSE	
4	"ParamDB".putData	Bool	FALSE	
5	"ParamDB".getData	Bool	FALSE	
6	"ParamDB".autoSend	Bool	FALSE	
7	"ParamDB".KAInterval	Time	T#3S	
8	"ParamDB".timeout	Time	T#500MS	
9	// Output parameters LBC_Manager			
10	"ParamDB".valid	Bool	FALSE	
11	"ParamDB".done	Bool	FALSE	
12	"ParamDB".busy	Bool	FALSE	
13	"ParamDB".error	Bool	FALSE	
14	"ParamDB".statusID	DEC+/-	0	
15	"ParamDB".status	Hex	16#0000	
16	"ParamDB".savedStatus	Hex	16#0000	
17	"ParamDB".connEst	Bool	FALSE	
18	"ParamDB".isMaster	Bool	FALSE	
19	"ParamDB".timeStats.meanTime	Time	T#0MS	
20	"InstLBC_Manager".statMode	DEC	0	
21	// Active slaves			
22	"ParamDB".numActSlaves	DEC+/-	0	
23	"ParamDB".slavesArray.slaves[0].isActive	Bool	FALSE	
24	"ParamDB".slavesArray.slaves[1].isActive	Bool	FALSE	
25	"ParamDB".slavesArray.slaves[2].isActive	Bool	FALSE	
26	"ParamDB".slavesArray.slaves[0].hasSentAck	Bool	FALSE	
27	"ParamDB".slavesArray.slaves[1].hasSentAck	Bool	FALSE	
28	"ParamDB".slavesArray.slaves[2].hasSentAck	Bool	FALSE	
29	// Data			
30	"ParamDB".generateData	Bool	FALSE	
31	"ParamDB".data.chars[0]	Character	' '	
32	"ParamDB".data.chars[1]	Character	' '	
33	"ParamDB".slavesArray.slaves[0].receivedData.data.chars[0]	Character	' '	
34	"ParamDB".slavesArray.slaves[0].receivedData.data.chars[1]	Character	' '	
35	"ParamDB".slavesArray.slaves[1].receivedData.data.chars[0]	Character	' '	
36	"ParamDB".slavesArray.slaves[1].receivedData.data.chars[1]	Character	' '	
37	"ParamDB".slavesArray.slaves[2].receivedData.data.chars[0]	Character	' '	
38	"ParamDB".slavesArray.slaves[2].receivedData.data.chars[1]	Character	' '	
39	// Slaves			
40	"ParamDB".slavesArray.slaves[0].address.IP.ADDR[4]	DEC	0	
41	"ParamDB".slavesArray.slaves[0].receivedData.receivedPutGet	DEC	0	
42	"ParamDB".slavesArray.slaves[0].receivedData.receivedKeepAlive	DEC	0	
43	"ParamDB".slavesArray.slaves[1].address.IP.ADDR[4]	DEC	0	
44	"ParamDB".slavesArray.slaves[1].receivedData.receivedPutGet	DEC	0	
45	"ParamDB".slavesArray.slaves[1].receivedData.receivedKeepAlive	DEC	0	
46	"ParamDB".slavesArray.slaves[2].address.IP.ADDR[4]	DEC	0	
47	"ParamDB".slavesArray.slaves[2].receivedData.receivedPutGet	DEC	0	
48	"ParamDB".slavesArray.slaves[2].receivedData.receivedKeepAlive	DEC	0	

7.2.2 Generating test data

Control the „LBC_ParamDB”.generateData tag to TRUE to have the „LBC_ParamDB”.data array filled with characters from “a” to “z”. Depending on the cycle time of the organization block in which “LBC_GenerateData” is called, the elements are overwritten with new characters during each cycle.

7.2.3 Sending a send frame

Table 7-3

No.	Action	Notes
1.	Make sure that the respective station is active as the master and no error is pending.	„LBC_ParamDB”.isMaster and „LBC_ParamDB”.valid are TRUE.
2.	Control the tags „LBC_ParamDB”.data.chars[0] and data.chars[1] to any character, or have them filled in automatically (see 7.2.2).	
3.	Control the „LBC_ParamDB”.sendData tag to TRUE.	A send frame is now sent with the data from the #data array. As soon as the frame was sent, #done is set for one cycle. Receiving the frame is acknowledged by the slaves and indicated at the #hasSentAck tag until the next send job.

7.2.4 Sending request frame

Table 7-4

No.	Action	Notes
1.	Make sure that the respective station is active as the master and no error is pending.	„LBC_ParamDB”.isMaster and „LBC_ParamDB”.valid are TRUE.
2.	Control the „LBC_ParamDB”.requestData tag to TRUE.	A request frame is now sent and data requested from the slaves. As soon as the frame has been sent, „LBC_ParamDB”.done is set for one cycle. The slaves now reply to the receiving of the frame by sending the data from their #data array. If these were not changed since sending the send frame, the slaves return the received data. The received data of the slaves are displayed in the elements of the „LBC_ParamDB”.slavesArray array.

7.3 Diagnostics

Function block “LBC_Main” outputs status messages at the #status output. Any detected error is specified in this way. In addition, the source of the error is specified at the #statusID output.

Table 02/2016-5

Status ID	Status	Meaning	Remedy
0	16#0000	No messages are pending.	
1	16#7001	Main: no master is active in the network	Enable the master function at a station.
1	16#8001	Main: the station shall be the master, however, a master is already active.	Check the configuration of your nodes, and ensure that only one master exists in the network.
1	16#8201	Main: value at input #KAInterval is smaller than 200 ms.	Hand over a keep-alive interval larger than 200 ms.
1	16#8202	Main: value at input #timeout is smaller than 100 ms.	Hand over a timeout time larger than 100 ms.
2	16#7400	SlavesManager: no slaves accessible	
2	16#7401	SlavesManager: new slave accessible	
2	16#7402	SlavesManager: timeout when waiting for a response from at least one slave.	Check the following settings, if several slaves are accessible: • Are all slaves physically connected with the network? • Has the user program been loaded in all slaves and are these slaves in RUN mode? • Are all slaves accessible via the network?
2	16#7403	SlavesManager: an inactive slave can be accessed again.	
2	16#7404	SlavesManager: the slave array is filled. No further slaves are managed.	Modify the maximum number of slaves to be managed in data type “LBC_typeSlavesArray” and download
3	16#xxxx	Error in “LBC_EstSockets”: status of TCON or TDISCON	Use the STEP 7 online help.
4	16#xxxx	Error in “LBC_ReceiveManager”: status of TUSEND or TURCV.	Use the STEP 7 online help.
5	16#xxxx	Error in “LBC_SendManager”: status of TUSEND.	Use the STEP 7 online help.
6	16#xxxx	Error in “LBC_SlaveManager”: status of TURCV.	Use the STEP 7 online help.

8 Related literature

Table 8-1

	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Download page of this entry https://support.industry.siemens.com/cs/ww/en/view/20983558

9 History

Table 9-1

Version	Date	Modifications
V1.0	05/2004	First version
V2.0	12/2015	Recreating the application example
V2.1	02/2016	Limit of the broadcast to the master’s subnet