SIEMENS

SIMATIC

Loadable Driver for CP 341
Modbus ASCII Master with 32-Bit Extensions

Manual

SIEMENS

SIMATIC

Loadable Driver for CP341 Modbus Protocol ASCII Format with 32-Bit Extensions S7 is Master

Manual

Preface, Contents

Product Description 1

Installation 2

Commissioning Driver 3

Transmission Protocol 4

Interface CPU - CP 5

Diagnostics Driver 6

Application Example 7

Appendices

Wiring Diagrams Multipoint A

Literature List B

Glossary

Edition 1.0

Safety Precautions and Warnings

This manual contains warnings, which you should note for your own safety as well as for the prevention of damage to property. These warnings are indicated by means of a triangle and displayed as follows in accordance with the level of danger:

Danger

indicates that death, severe personal injury or substantial damage **will** result if proper precautions are not taken.

Warning

indicates that death, severe personal injury or substantial damage **can** result if proper precautions are not taken.

Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken.

Notice

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Qualified Personnel

The equipment may be commissioned and put into operation by **qualified personnel** only. For the purpose of safety relevant warnings of this manual a qualified person is one who is authorized to commission, ground and tag devices, systems and circuits.

Correct Usage

Please note the following:

Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Trademarks

SIMATIC® and SINEC® are registered trademarks of SIEMENS AG.

The other brand names in this manual may be trademarks use of which by third parties for their purposes may infringe the proprietors' rights.

Disclaimer of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcome.

Preface

Purpose of this Manual

The information in this manual will enable you to establish and commission a data link between a CP 341 and a “Modbus capable” control system.

Required Basic Knowledge

You require a general knowledge in the field of automation engineering to be able to understand this manual.
In addition, you should know how to use computers or devices with similar functions (e.g. programming devices) under Windows 95/98/2000/NT or XP operating systems. Since loadable driver are based on the STEP 7 software, you should also know how to operate it. This is provided in the manual "Programming with STEP 7 V5.2".

Contents of the Manual

This manual describes the loadable driver functions and how to create a link to the hardware and software of communication processor CP 341.

The manual contains the following subjects:

- Product Description / Installation
- Commissioning the Driver / Installation / Parameterization
- Interface CPU-CP
- Transmission Protocol
- Diagnostics Driver
- Application Example

Validity of the Manual

This manual Issue is valid for the following software package:

Product	Identification No.	from Version
Loadable Driver for CP 341 Modbus ASCII Master	6ES7870-1CA00-0YA0	1.0

Note

This manual contains the driver description as is valid at the time of publication.

How to access the information in this manual

To enable you to access the information in this manual more easily, we would like to draw your attention to the following:

- The next few pages contain a complete list of contents.

Further sources of information

Any further information regarding CP 341 (installation, commissioning etc.) can be found in the following manual:

SIEMENS
SIMATIC
CP341 Point to Point Communication
Installation and Parameter Assignment
Manual
C79000-G7076-C341-..

Further information regarding STEP7 can be found in the following manuals:

SIEMENS
SIMATIC Software
Standard Software for S7 and M7
STEP7 User Manual
C79000-G7000-C502-..

SIEMENS
SIMATIC Software
System Software for S7-300/400
System- and Standard Functions
Reference Manual
C79000-G7000-C503-..

Queries

Should you have any queries regarding the use of the driver described in this manual, which are not answered in this documentation please contact the relevant person at Siemens who supplied you with this driver.

Terminology

This documentation uses the terms CP or CP341.

Scope of Application

The driver described in this manual serves as a loadable protocol for CP341, which may be used instead of Standard Protocols 3964R, RK512, and ASCII.

Note

With this driver, modifications or expansions to the sequences between CP and CPU are possible.

These modifications and expansions may apply in particular to event classes or event numbers available for diagnostic purposes.

Furthermore please note that this manual only describes the modifications and expansions as against the standard functions. Basic information may be found in the manuals mentioned in chapter “Further Sources of Information“.

In order to ensure safe use of the driver, detailed knowledge of the functionality of CP341 is a pre-requisite.

Contents

1 Product Description 1-1

1.1 Usage Possibilities 1-1

1.2 Hardware and Software Prerequisites 1-2

1.3 Summary of the Modbus Protocol 1-2

2 Installation 2-1

2.1 Use of the Dongle 2-1

2.2 Interface Connection 2-1

3 Commissioning the Driver 3-1

3.1 Installation of the Driver on the STEP7-PG/-PC 3-1

3.2 Uninstalling the Driver 3-2

3.3 Configuring a Data Link CP in Step7 3-2

3.4 Assigning Parameters to the CP 3-3

3.5 Loading the Driver to the CP 3-4

3.6 Assigning Parameters to the Loadable Driver 3-4

3.6.1 Modbus ASCII Protocol 3-5

3.6.2 RS422/485 (X27) Interface 3-7

3.6.3 RS232 Secondary Signals 3-8

3.7 Loading the Configuration and Parameter Assignment Data 3-9

3.8 Start-up Characteristics of CP341 3-9

4 Transmission Protocol 4-1

4.1 Message Structure 4-1

4.2 Exception Responses 4-5

4.3 RS 232C Secondary Signals 4-6

5 Function Codes 5-1

5.1 Function Code 01 – Read Coils........ 5-2

5.2 Function Code 02 – Read Discrete Input 5-3

5.3 Function Code 03 – Read Holding Registers 5-5

5.4 Function Code 03 – Read 32-Bit Holding Registers........ 5-6

5.5 Function Code 04 – Read Input Registers 5-7

5.6 Function Code 05 – Write Single Coil........ 5-8

5.7 Function Code 06 – Write Single Register 5-9

5.8 Function Code 06 – Write Single 32-Bit Register........ 5-10

5.9 Function Code 07 - Read Exception Status 5-11

5.10 Function Code 08 – Diagnostics (Loop Back Test)........ 5-12

5.11 Function Code 11 – Get Comm Event Counter........ 5-13

5.12 Function Code 12– Get Comm Event Log........ 5-14

5.13 Function Code 15 – Write Multiple Coils 5-15

5.14 Function Code 16 – Write Multiple Registers 5-16

5.15 Function Code 16 – Write Multiple 32-Bit Registers........ 5-17

6 CPU – CP Interface........ 6-1

6.1.1 Data Transfer from CPU to CP with P_SND_RK 6-1

6.1.2 Data Transfer from CPU to CP with P_RCV_RK 6-4

7 Diagnostics of the Driver........ 7-1

7.1 Diagnostics via Display Elements (LEDs) 7-1

7.2 Diagnostic Messages of the Function Blocks........ 7-2

7.3 Table of Errors / Events........ 7-3

7.3.1 Error Codes for “CPU Job Errors” 7-3

7.3.2 Error Codes for “Receive Errors”........ 7-4

7.3.3 Error Codes for “General Processing Errors” 7-6

8 Application Example 8-1

8.1 Used Blocks 8-1

8.1.1 Program Description 8-3

8.1.2 Programming Example 8-4

A Wiring Diagrams Multipoint 1

B Literature List 1

Glossary 1

1 Product Description

1.1 Usage Possibilities

Position in the System

The driver described here is a software product for communication processor CP341.

Environment

CP341 can be used in automation systems S7-300 and can establish serial communication links to partner systems.

Function of the Driver

This driver enables you to establish a communication link between communication module CP341 and “Modbus capable” slaves.

The transmission protocol used is the **Modbus Protocol** in **ASCII Format**. In addition, de-facto standard 32-bit extensions are supported for accessing floating point and double-word registers in compatible slaves. Data transmission is carried out in accordance with the Master-Slave principle. The **Master (SIMATIC S7)** has the initiative during the transmission.

Function codes **01, 02, 03, 04, 05, 06, 07, 08, 11, 12, 15, and 16** can be used for communication between the CP and the slaves.

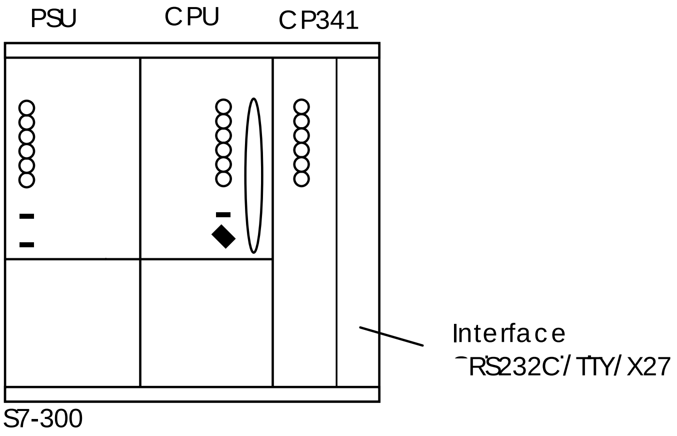
Usable Interfaces and Protocols

You can use this driver on a CP341 having a RS232, TTY, or RS422/485 (X27) interface.

With this driver, it is possible to use the RS422/485 (X27) interface submodule in both 2-wire operation and 4-wire operation. In 2-wire operation. It is possible to connect up to 32 slaves to one master in half-duplex operation, thus creating a multipoint connection (network).

Possible System Configuration

Please find below an illustration of system configuration schematics.



1.2 Hardware and Software Prerequisites

Useable Module

The Driver runs on CP341 with part number 6ES7 341-1AH01-0AE0 as well as -1BH01 and -1CH01. Also the previous modules -1AH00, -1BH00 and -1CH00 can be used with this driver.

Dongle

In order to use the CP with loadable drivers, you require a dongle. The dongle with identification number 6ES7870-1CA00 is supplied with the driver.

Loading Memory of the CPU (Memory Card)

Every CP interface, for which this loadable driver has been assigned parameters, requires a CPU loading memory amount of about 25 Kbytes.

With CP 341 the loadable drivers are downloaded directly to the CP 341. Therefore you do not require a loading memory on the S7-300 CPU. You should note, however, that this means that you cannot change a module without a programming device.

Software Issue Levels

Loading of drivers is possible with **STEP 7** from issue level 4.04.

An installed version of the **Parameter Assignment Tool** *CP: Point-to-Point Communication, Parameter Assignment* V4.1 or higher is required.

We recommend to use STEP 7 V5.1 or higher and Parameter Assignment Tool V5.1 or higher.

Data Structures

Prior to project configuration of your S7 data structures, you should ensure that they are compatible with the user programs of the Modbus Slave systems (clarify which function codes and which Modbus addresses will be used).

1.3 Summary of the Modbus Protocol

Function Codes

The type of data exchange between Modbus systems is controlled by Function Codes (FCs).

Data Exchange

The following FCs can be used to carry out data exchange **bit-by-bit**:
FC 01 Read Coils,
FC 02 Read Discrete Inputs,
FC 05 Force Single Coil,
FC 15 Force Multiple Coils.

The following FCs can be used to carry out data exchange **register-by-register**:
FC 03 Read Holding registers,
FC 04 Read Input registers,
FC 06 Write Single Register,
FC 16 Write Multiple Registers.

Data Areas

As a rule, the individual FCs operates in accordance with the table below:

Function Code	Data	Type of Data		Access
01, 05, 15	Coil (output) status	Bit	Output	read/write
02	Input Status	Bit	Input	read only
03, 06, 16	Holding Register	Register 16 or 32 bit	Output Register	read/write
04	Input Register	Register 16 or 32 bit	Input Register	read only

Address Representation

Analogous to the partitioning into read/write and read-only areas, data at user level can be represented as shown in the table below:

Function Code	Type of Data	Example: Address Representation at User Level (Decimal)
01, 05, 15	Output bit	0xxxx
02	Input bit	1xxxx
04	Input register	3xxxx
03, 06, 16	Holding register	4xxxx

In the **transmission messages** on the serial transmission line, the addresses used in the Modbus user system are referenced to **0**.
In the **Modbus user system** itself, these addresses are counted beginning with **1**.

Example:
If the first holding register in the user system is represented as register **4**0001, in the transmission message the value 0000 Hex is transmitted as the register address when FC 03, 06, or 16 is used to access register **4**0001
If the 127th coil is represented as coil **0**0127 in the user system, it is assigned the coil address 007E Hex in the transmission message.

Note:
The CP341 driver only deals with the transmitted or received zero-based PDU addresses. Any translation from the user level address must be handled in the application program in the S7 PLC or the associated HMI.

2 Installation

2.1 Use of the Dongle

Introduction

In order to run the CP with loadable drivers, you require a dongle. When the dongle is plugged in, drivers can be loaded.

How to Plug In the Dongle

Before you can plug in the dongle, you must take the CP out of the rack. At the back of the CP, above the plugs for the backplane bus, there is a slot into which the dongle can be inserted.

2.2 Interface Connection

TTY

A point-to-point connection to one slave can be realized.

Further notes to the interface connection please find in the manual “CP341 Point to Point Communication“.

RS232C

A point-to-point connection to one slave can be realized. It is possible to use RS232 auxiliary signals for, e.g., modem control.

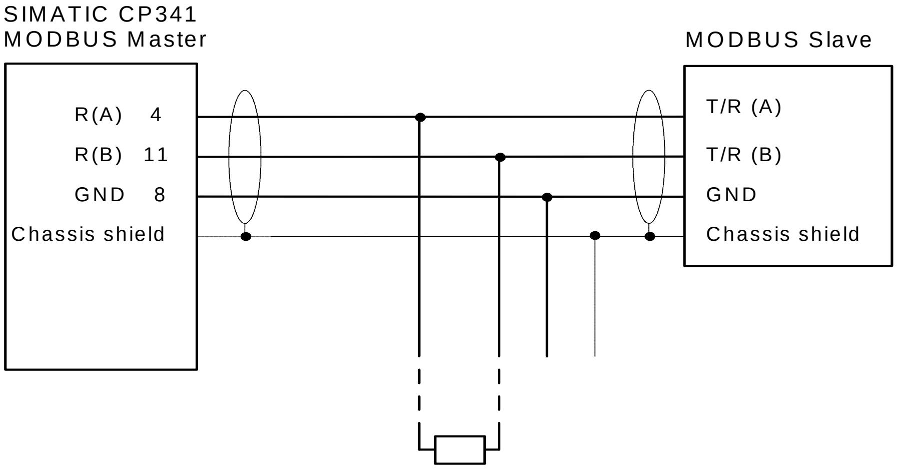
Further notes to the interface connection please find in the manual “CP341 Point to Point Communication“.

X27 (2-wire)

A multipoint connection (network) connecting up to 32 slaves to one Master can be created directly.

The driver of the CP performs the switchover of the receive-2-wire line between transmit and receive.

Schematic connection: 1 Master system, 1 slave at the bus



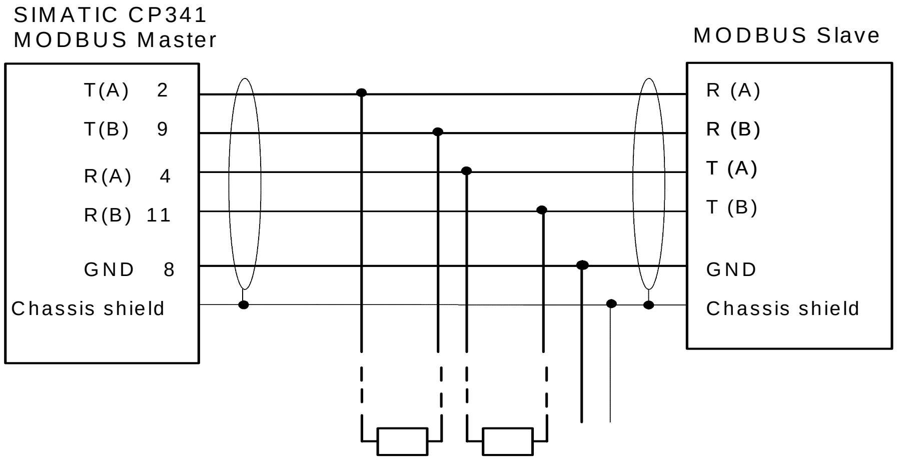
Further notes to the interface connection please find in the manual “CP341 Point to Point Communication“.

X27 (4-wire)

A Point-to-Point connection to one slave can be created.

The direct construction of a multipoint connection (network) connecting more than one slave is not possible.

Schematic connection: 1 Master system, 1 Slave



Further notes to interface connection please find in the manual “Point-to-Point Data Link CP341“.

3 Commissioning the Driver

General Information

All statements in the following sections referring to STEP7 or configuring or setting parameters for CP-PtP, CP341 or the Driver are related to the STEP7-Version 5.3 SP3.

Operation flows, names and directory names might be different in other STEP7 versions.

3.1 Installation of the Driver on the STEP7-PG/-PC

Prerequisites

To make the driver installation possible, a **STEP7-Package** and the **Parameter Assignment Tool** *CP: Point-to-Point Communication, Parameter Assignment* must have been installed before.

Installation

Installation of the driver consisting of driver code and driver specific configuration screens for STEP7. Insert your Modbus ASCII Driver CD into the CD-ROM drive and follow step-by-step the instructions that are automatically displayed by the installation program. If the installation program fails to automatically run, perform these steps:

1. Using Windows Explore, navigate to the CD-ROM drive and go to the directory MODBUS_ASCII_MASTER and double-click **Setup.EXE** file to start the installation procedure.
2. Follow step-by-step the instructions that are displayed by the installation program.

Result: The driver and the parameterization masks are installed in the following directory: [c:\Program Files\]**Step7\S7fptp\S7Driver** where the contents of [] are selectable during the installation procedures

The directory includes the following files:

- S7wfpmab.dll
- S7wfpmax.cod
- S7wfpmbx.cod

3.2 Uninstalling the Driver

The driver can be uninstalled from the STEP 7 package by selecting “Control Panel”, “Add / Remove Software” Find the driver in the list and follow the instruction for uninstalling it.

The user can check if all the files S7wfpma?.*, S7wfpmb?.*, S7wfpmc?.* have been deleted successfully in the [c:\Program Files\]Step7\S7fptp\S7Driver directory.

Note:
Before uninstalling the package “**Parameter Assignment Tool** *CP: Point-to-Point Communication, Parameter Assignment“* all the loadable drivers must first be uninstalled.

3.3 Configuring a Data Link CP in Step7

Introduction

The configuration of a data link comprises the hardware allocation in the configuration table using HW config. The configuration can be carried out using the STEP 7 software.

S7-Project

Before you can carry out the configuration, you must have created a **S7 Project** with STEP 7.

Project Components

Insert the required project components into the opened project using the SIMATIC Manager. You must have a “SIMATIC 300 Station” in your project.

Before an insertion, you must select the target project name by clicking it. To insert the 300 Station, from the Insert menu of Simatic Manager do:

Insert → Station → SIMATIC 300 Station

Hardware Configuration

The configuration of the hardware comprises defining the hardware components themselves, and also their properties.

To start the hardware configuration, select the SIMATIC 300 Station and double-click “Hardware” (or select the menu command **Edit → Open Object**). Use the menu command **Insert → Hardware Components** to insert a RACK- 300, a PS-300, a CPU-300 from SIMATIC 300, and the CP PtP from CP-300 with the appropriate part number.

A detailed description of how to configure S7-300 modules can be found in the User Manual for STEP 7.

3.4 Assigning Parameters to the CP

Assigning Parameters to the CP

After you have arranged the modules in your rack using “Hardware Configuration,” you must assign parameters to them.

To start the parameter assignment tool, double-click the CP in “Hardware Configuration” or click the CP and select the menu command **Edit** ➢ **Object Properties.**

1) <u>Properties - CP341 ➢ Basic Parameters Tab</u>

   Clicking the **“Parameter…**” button along the bottom opens the protocol selection interface **“Parameter Assignment of Point-to-Point Connection**”. Here you can select the required driver protocol, **Modbus ASCII Master,** from the drop-down menu.

   After selecting the **Protocol,** you can carry out the **Parameter Assignment of the Driver** (start by double-clicking the envelope symbol labeled “Protocol”).

   A detailed description of how to select the protocol and assign parameters to the dialog boxes for the loadable driver can be found in the section “Assigning Parameters to the Loadable Driver.”

   After parameter assignment is complete, you return to the **“Properties - CP**” dialog box.

2) <u>Properties - CP341 ➢ Addresses</u>

   No settings are required in the **“Addresses**” tab (Properties - CP dialog box).

3) <u>Properties - CP341 ➢ General</u>

   No settings are required in the **“Genera**l” tab (Properties - CP dialog box).

   You can complete the parameter assignment of the CP by clicking “OK” in the “Properties - CP” dialog box. You then return to the “Hardware Configuration” dialog box.

   Save the parameter assignment and close the “Hardware Configuration” dialog box. You return to the basic menu of the STEP 7 project“.

3.5 Loading the Driver to the CP

Loading the Driver

After selection of a loadable driver in the selection box “Protocol”, you can load the driver to the CP. Double clicking on to the icon **“Load Drivers”** gets you to the dialogue where the driver is loaded.

- You need an **online** connection to the CPU to load drivers.
- The tab “Load Drivers” shows you, which driver is already loaded on the CP and which driver was selected by you.
- Once again click “Load Drivers” and confirm with “yes”. The transfer of the driver to the CP is carried out.
- After the transfer the information “Driver version online on the module” is updated.
- If the driver in the current version already exists on the CP, the transfer in cancelled with the message “Driver already exists”.
- Click “Close” to return to the main tab.

The error “Module rejected driver download” may occur, when the driver files are destroyed. In that case a re-installation of the driver is necessary.

3.6 Assigning Parameters to the Loadable Driver

Opening the Parameter Assignment Tool CP-PtP

Select the SIMATIC station and double-click “Hardware” (or select the menu command **Edit → Open Object**) to start the “Hardware Configuration.”
Click the CP and select the menu command **Edit → Object Properties**. Click the “**Parameter…**” button along the bottom to open the protocol selection dialog box.

Protocol Selection

In addition to the standard protocols the selection box also displays all installed loadable drivers. Choose “**Modbus ASCII Master**“ for this loadable driver. Double clicking on to the symbol for the transmission protocol (an envelope icon) gets you to the dialogue where the protocol specific parameters are set.

Driver-Specific Parameters

The parameters described below can be set for this driver in the individual dialog boxes.

3.6.1 Modbus ASCII Protocol

Overview of Transmission Parameters

Transmission Parameters			
Parameter	**Description**	**Value range**	**Default value**
Baud rate	Data transmission speed in bits / second	300 600 1200 2400 4800 9600 19200 38400 76800	9600
Data bits	Bit per character	7	7
Stop bits	Amount of stop bits	1 2	1
Parity	amount of data bits is completed to an even number amount of data bits is completed to an odd number no parity bit transferred	even odd none	even

Transmission Rate

The transmission rate is the speed of data transmission in bits per second (bps).

Data Bits

The amount of data bits describes how many bits represent a character to be transmitted. With Modbus ASCII 7 data bits are mandatory.

Stop Bits

The amount of stop bits defines the smallest possible distance between two characters to be transferred. With even or odd parity 1 stop bit is pre-defined. None parity effects two stop bits.

Parity

The parity bit is for data safety; depending on parameter assignment, it completes the amount of transmitted data bits to either an even or an odd number.

If “no” parity is selected, no parity bit is transmitted. This reduces the safety of data transmission.

Overview of Protocol Parameters

Protocol Parameters			
Parameter	**Description**	**Value range**	**Default value**
Character delay time	Time period used to monitor the incoming characters within a message	1 to 6500 milliseconds in 1ms intervals	1000ms
Response time-out	Time to monitor the start of the reply from the slave	5 to 65500 milliseconds in 1ms intervals	2000
Turnaround delay	Waiting time after sending a broadcast message		
Operating mode	"Normal Operation" "Interference Suppression"	Normal Interference Suppression	Normal
32-Bit mode	Registers can also imply 32-bit values	not selected selected	not selected

Character Delay Time

When receiving a message the quiet time between characters is measured. If the quiet time exceed the the character delay time, the message is ignored and an error is reported.

Response Time-out

The reply monitoring time is the time the master spends waiting for a reply message from the slave after output of a request message. If the start character is not received within the response timeout, the message is ignored and an error is reported.

Turnaround Delay

When a broadcast request is sent, no response is returned from the slaves. Nevertheless a delay is respected by the Master in order to allow any slave to process the current request before sending a new one. This delay is called turnaround delay. The turnaround delay should be shorter than the response timeout.

After the CP has sent the last character of a broadcast message it waits the turnaround delay before the send job is completed. If the turnaround delay is set to 0, the CP completes the send job immediately after sending the last character of the request.

Reply Monitoring Time

The reply monitoring time is the time the master spends waiting for a reply message from the slave after output of a request message.

If the slave doesn't send a start character during the reply monitoring time the send job is finished with error.

Normal Operation

In this operating mode, all recognized transmission errors and/or BREAK before and after receive messages from the slave result in an appropriate error message.

Interference Suppression

If “BREAK” is recognized on the receiving line at the start of the receive message, or if the CP interface block notices transmission errors before the message, no error is reported.

The start of the receive message from the slave is recognized by means of the correctly-received start character. Transmission errors and/or BREAK are also ignored when they occur after the end of the receive message.

32-Bit Mode

Normally registers are 16-bit values. When choosing 32-bit mode, registers can also imply 32-bit values when supported in the addressed slave.

3.6.2 RS422/485 (X27) Interface

Overview

X27 (RS 422/485) - Interface Sub-module			
Parameter	**Description**	**Value range**	**Default value**
Presetting of the receiving line	No presets Preset “Break“ Preset “High“	none R(A)5V,R(B)0V R(A)0V,R(B)5V	R(A)5V, R(B)0V
X27-Operation mode	Via the transmission line T(A), T(B) data are sent, via the receiving line R(A), R(B) data are received. The receiving line R(A),R(B) is changed-over from send to receive operation.	Full-duplex / four-wire-operation Half-duplex / two-wire-operation	Full-duplex / four-wire-operation

"Full-duplex / four-wire-operation“

In this operating mode, data are sent via the transmission line T(A),T(B) and received via the receiving line R(A),R(B). Error handling is carried out in accordance with the function set at the “Driver Operating Mode” parameter (Normal or Interference Suppression).

"Halfduplex / two-wire-operation“

In this operating mode, the driver **switches** the 2-wire receiving line R(A),R(B) of the interface from send to receive operation. In this operating mode, all recognized transmission errors and/or BREAK before and after receive messages are ignored. BREAK level during message pauses is also ignored. The beginning of the receive message from the slave is recognized by means of the correctly-received colon character.

The setting R(A) 0V, R(B) 5V (High) is recommended as the preset for the receiving line.

Presetting of the Receiving Line

“None“ (Float)

The two-wire-line R(A),R(B) is **not** preset.
In this instance the link partner should carry out assignment.

Presetting “R(A) 5V, R(B) 0V“ (BREAK)

The two-wire-line R(A),R(B) is preset by the CP as follows:
R(A) --> +5V, R(B) --> 0V (V_A - V_B $\geq$ +0,3V).
This means that BREAK level occurs on the CP in the event of a line break.

Presetting “R(A) 0V, R(B) 5V“ (High)

The two-wire-line R(A),R(B) is preset by the CP as follows:
R(A) --> 0V, R(B) --> +5V (V_A - V_B $\leq$ -0,3V).
This means that HIGH level occurs on the CP in the event of a line break (and / or when it is running idle, i.e. no slave is transmitting).
Line status BREAK cannot be recognized.

3.6.3 RS232 Secondary Signals

Overview

Data Transmission			
Parameter	**Description**	**Value range**	**Default value**
Automatic use of RS232 signals	RS232 secondary signals are enabled	yes no	no
Time to RTS OFF	Time to elapse after the transmission before the CP sets the RTS line to OFF	0 to 655350 ms in 10 ms steps	1s
Data output waiting time	Delay before the CP starts sending of a telegram	0 to 655350 ms in 10 ms steps	1s

Automatic Use of RS232 Signals

With this parameter you can choose whether RS 232 C secondary signals are used or not. If no secondary signals are parameterized, the CP neither sets nor checks these signals.

The description of the used secondary signal please find in Chapter 4 of this manual.

Time to RTS OFF

After output of a request the CP waits the defined time to set the RTS line to OFF.

Data Output Waiting Time

The data output waiting time is the time that the CP 341 is to wait for the communication partner to set CTS to ON after setting the RTS line to ON and before starting the transmission.

Selecting Parameters

Select the parameters required for your data link and exit the individual dialog boxes by clicking “OK“.

3.7 Loading the Configuration and Parameter Assignment Data

Data Management

On closing the “Hardware Configuration“ the data is automatically saved into your Step7-project.

Loading of Configuration and Parameters

The configuration- and parameterization data can now be downloaded online from the programming unit to the CPU. Use menu commands **PLC ➢ Download** to transfer the data to the CPU.

During CPU startup and each time you switch between STOP mode and RUN mode, the module parameters of the CP are automatically transferred to the CP as soon as it can be reached via the S7-300 backplane bus.

The driver code is not saved in the CPU, but directly with the parameter assignment tool in the retentive memory of the CP 341. You should note, however, that this means you cannot change a module without a programming device.

Further Information

Please refer to the User Manual for STEP7 for detailed description on:

- how to save the configuration and the parameters.
- how to load the configuration and the parameters into the CPU.
- how to read, change, copy and print the configuration and the parameters.

3.8 Start-up Characteristics of CP341

Introduction

The startup of the CP is divided into two phases:
––> Initialization (mains-on of CP)
––> Parameter assignment

Initialization

As soon as voltage is applied to the CP, and after completion of a hardware test program, the firmware on the CP is prepared for operation.

Parameter Assignment

During parameter assignment, the CP receives the module parameters allocated to the current slot. The CP is now ready to run.

4 Transmission Protocol

General Information

The procedure in use is asynchronous and half-duplex. Data transmission is carried out without handshake.

Master-Slave Relationship

The CP initiates the transmission (= Master), and after outputting a request message it waits for a reply message from the slave for the duration of the parameter “reply monitoring time” .

ASCII-Mode

When devices are setup to communicate on a Modbus serial line using ASCII mode, each 8–bit byte in a message is sent as two ASCII characters.

The allowable characters transmitted for all fields except the start character and end characters are hexadecimal 0–9, A–F (ASCII coded).

Example: The byte 0X5B is encoded as two characters: 0x35 and 0x42 (0x35 ="5", and 0x42 ="B" in ASCII).

4.1 Message Structure

Message Structure

The data exchange “Master-Slave” and/or “Slave-Master” begins with the **Start Character**, followed by **Slave Address** and **Function Code**. Then the data are transferred. The structure of the data field depends on the function code used. The LRC check is transmitted at the end of the message, followed by the **End Characters**.

START	ADDRESS	FUNCTION	DATA	LRC	END
1 char colon	2 chars	2 chars	0 up to 2x252 char(s)	2 chars	2 chars CR, LF

START	Start Character :
ADDRESS	Modbus Slave Address
FUNCTION	Modbus Function Code
DATA Message	Data: Byte_Count, Coil_Number, Data
LRC	Message Checksum
END	End Characters CR, LF

Start Character

The start character is a colon (0x3A). The devices monitor the bus continuously for the ‘colon’ character. When this character is received, each device decodes the next character until it detects the End Characters (CR,LF).

Slave Address

The slave address can be within the range 1 to 255. The address is used to address a defined slave on the bus.

Broadcast Message

The master uses slave address zero to address all slaves on the bus. **Broadcast Messages** are only permitted in conjunction with writing **Function Codes 05, 06, 15, and 16**. A Broadcast Message is not followed by a reply message from the slave.

After a broadcast message the CP waits for a time determined by the "turnaround delay" parameter before the send job is finished.

Function Code

The function code defines the meaning as well as the structure of a message. The following function codes are supported by the driver:

Function Code	Function in Accordance with Modbus Specification
01	Read Coils
02	Read Discrete Inputs
03	Read Holding Registers
04	Read Input Registers
05	Write Single Coil
06	Preset Single Register
07	Read Exception Status
08	Loop Back Diagnostic Test
11	Fetch Communications Event Counter
12	Fetch Communications Event Log
15	Write Multiple Coils
16	Write Multiple Registers

Data Field DATA

The data field DATA is used to transfer the function code-specific data such as: Bytecount, Coil_Start Address, Register_Start Address; Number_of_Coils, Number_of_Registers, ... See also Chapter "Function Codes".

The data field contains up to 2x252 ASCII characters.

LRC

The Longitudinal Redundancy Checking (LRC) field is one byte, containing an 8–bit binary value. The LRC value is calculated by the transmitting device, which appends the LRC to the message. The device that receives recalculates an LRC during receipt of the message, and compares the calculated value to the actual value it received in the LRC field. If the two values are not equal, an error results.

The LRC is calculated by adding together successive 8–bit bytes in the message, discarding any carries, and then two's complementing the result. The LRC is an 8–bit field, therefore each new addition of a character that would result in a value higher than 255 decimal simply 'rolls over' the fields value through zero. Because there is no ninth bit, the carry is discarded automatically.

A procedure for generating an LRC is:

1. Add all bytes in the message, excluding the starting ‘colon’ and ending CRLF. Add them into an 8–bit field, so that carries will be discarded.

2. Build the two’s–complement.

3. Convert the LRC to ASCII.

Placing the LRC into the Message

When the 8–bit LRC (2 ASCII characters) is transmitted in the message, the high–order character will be transmitted first, followed by the low–order character. For example, if the LRC value is 61 hex (0110 0001):

LRC high	0x36
LRC low	0x31

Message End

The end of the message is defined by the characters CR and LF.

Telegram Example

The Modbus serial line PDU is describes as follows:

05H	Slave Address
08H	Function Code
00H	Diagnostic Code “High”
00H	Diagnostic Code “Low”
A5H	Test Value “High”
C3H	Test Value “Low”
xxH	LRC

In ASCII transmission mode the following data is transferred on the line:

3AH	Start Character
30H	Slave Address
35H	
30H	Function Code
38H	
30H	Diagnostic Code “High”
30H	
30H	Diagnostic Code “Low”
30H	
41H	Test Value “High”
35H	
43H	Test Value “Low”
33J	
xxH	LRC Code High
xxH	LRC Code Low
0DH	CR
0AH	LF

Error Handling

If BREAK is recognized on the receiving line by the CP during output of a message, the triggered P_SND_RK job is completed with error. Reception during transmission is ignored.

If any of the errors listed below is recognized by the CP during reception of the reply message, the received data string is rejected, an error is reported and the triggered Send job is completed with error.

- reply monitoring time elapsed
- wrong start character
- received character is no ASCII character
- overrun of the receive buffer
- received LRC incorrect
- transmission error in a character (parity, framing or overrun error)
- character delay time elapsed
- BREAK (line break or DSR or CTS not asserted)

4.2 Exception Responses

Exception Responses

On recognition of an error in the request message from the master (for example, register address illegal), the slave sets the highest value bit in the function code of the reply message. This is followed by transmission of one byte of error code (Exception Code), which describes the reason for the error.

A detailed description of the meaning of the above-mentioned parameters can be found in the “Modbus Application Protocol Specification.”

Exception Code Message

The error code reply message from the slave has the following structure:
for example, slave address 5, function code 5, exception code 2

Reply Message from Slave EXCEPTION_CODE_xx:

05H	Slave Address
85H	Function Code
02H	Exception Code (1 to 7)
xxH	LRC

On receipt of an error code reply message by the driver, the current job is completed with error. An error number corresponding to the received error code (Exception Code 1-7) is also entered in the STATUS area. No entry is made in a P_RCV_RK destination data block.

The following error codes are defined in accordance with the Modbus Specification:

Exception Code	Meaning in accordance with Modbus Specification	Cause – Short Description *
01	Illegal Function	Illegal function code
02	Illegal Data Address	Slave has illegal data address
03	Illegal Data Value	Slave has illegal data value
04	Failure in Associated Device	Slave has internal error
05	Acknowledge	Function is carried out
06	Busy, Rejected Message	Slave is not ready to receive
07	Negative Acknowledgement	Function cannot be carried out

* Check slave for further details.Not all are supported by driver. See Modbus spec for detailed descriptions.

4.3 RS 232C Secondary Signals

Available Signals

The following RS 232C secondary signals exist on the CP when the RS232C interface submodule is used:

•	DCD	(input)	Data carrier detect; Data carrier detected
•	DTR	(output)	Data terminal ready; CP ready for operation
•	DSR	(input)	Data set ready; Communication partner ready for operation
•	RTS	(output)	Request to send; CP ready to send
•	CTS	(input)	Clear to send; Communication partner can receive data from the CP (response to RTS = ON of the CP)
•	RI	(input)	Ring indicator; Indication of an incoming call

When the CP is switched on, the output signals are in the OFF state (inactive).

You can parameterize the way in which the DTR/DSR and RTS/CTS control signals are used with the **CP 341: Point-to-Point Communication, Parameter Assignment** parameterization interface or control them by means of function calls (FBs) in the user program.

Using the RS 232C Secondary Signals

The RS 232C secondary signals can be used as follows:

- When the automatic use of all RS 232C secondary signals is parameterized
- By means of the V24_STAT and V24_SET functions (FBs)

Note
When automatic use of the RS 232C secondary signals is parameterized, neither RTS/CTS data flow control nor RTS and DTR control by means of the V24_SET FB are possible. On the other hand, it is always possible to read all RS 232C secondary signals by means of the V24_STAT FB.

The sections that follow describe how the control and evaluation of the RS 232C secondary signals is handled.

Automatic Use of the Secondary Signals

The automatic use of the RS 232C secondary signals on the CP is implemented as follows:

- As soon as the CP is switched by means of parameterization to an operating mode with automatic use of the RS 232C secondary signals, it switches the RTS line to OFF and the DTR line to ON (CP ready for use).
- Message frames cannot be sent and received until the DTR line is set to ON. As long as DTR remains set to OFF, no data is received via the RS 232C interface. If a send request is made, it is aborted with an error message.
- When a send request is made, RTS is set to ON and the parameterized data output waiting time starts. When the data output time elapses and CTS = ON, the data is sent via the RS 232C interface.
- If the CTS line is not set to ON within the data output time so that data can be sent, or if CTS changes to OFF during transmission, the send request is aborted and an error message generated.
- After the data is sent, the RTS line is set to OFF after the parameterized time to RTS OFF has elapsed. The CP does not wait for CTS to change to OFF.
- Data can be received via the RS 232C interface as soon as the DSR line is set to ON. If the receive buffer of the CP threatens to overflow, the CP does not respond.
- A send request or data receipt is aborted with an error message if DSR changes from ON to OFF. The message "DSR = OFF (automatic use of V24 signals)" is entered in the diagnostics buffer of the CP.

Note
When automatic use of the RS 232C secondary signals is parameterized, neither RTS/CTS data flow control nor RTS and DTR control by means of the V24_SET FB are not possible.

Note
The "time to RTS OFF" must be set in the parameterization interface so that the communication partner can receive the last characters of the message frame in their entirety before RTS, and thus the send request, is taken away. The "data out put waiting time" must be set so that the communication partner can be ready to receive before the time elapses**.**

Time Diagram

The following Figure illustrates the chronological sequence of a send request.

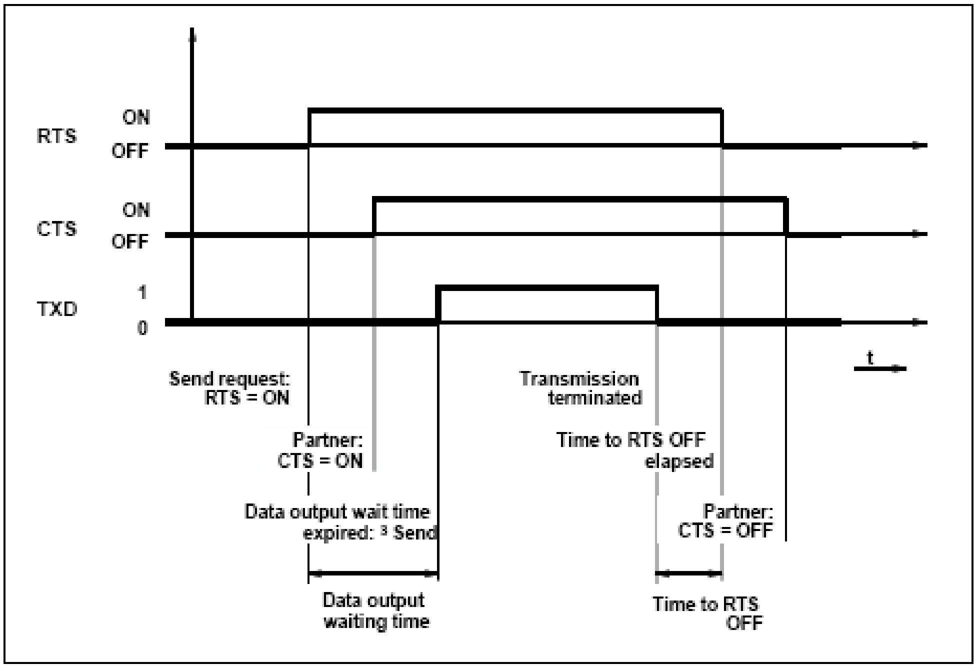


Figure 4-1 Time Diagram for Automatic Use of the RS 232C Secondary Signals

5 Function Codes

General

All telegram examples for the different function codes refer to Modbus serial line PDU format.

32-Bit Registers

The register oriented function codes 3, 6, 16 can also handle 32-bit registers. If Protocol Parameter for Modbus-Master “With 32-bit Registers” is set the driver is prepared to handle registers with the length of 4 byte.

The decision whether the send job refers to 16-bit or 32-bit registers is done via the second byte of the send data block. The second byte of the send data block determines the Modbus Function Code sent in the message. If bit 2^6 (the bit to the right of the most significant bit) is set, the send job refers to 32-bit registers. Bit 2^6 doesn’t affect the function code actually sent, it is just information for the master CP for what to expect in the response from the slave when reading or what to send when writing.

The register(s) accessed in the slave when bit 2^6 is set must be within a 32-bit register address range defined in the slave such that 4 bytes per register is returned in the read response or 4 bytes per register are expected in a write request.

If bit 2^6 is set and a normal 2 bytes per register range is read in the slave, 2 bytes per register is returned by the slave. As the master expects more data, the activated send job is finished with error. Likewise, bit 2^6 is not set and a 4 bytes per register range is read in the slave, 4 bytes per register is returned by the slave. The master receives more data than expected and the activated send job is finished with error.

If bit 2^6 is set and a normal 2 bytes per register range is written to the slave, 4 bytes per register are sent by the master and the slave should returned an exception response to the master. Likewise, if bit 2^6 is not set and a 4 bytes per register range is written to the slave, 2 bytes per register are sent by the master and the slave should returned an exception response to the master.

5.1 Function Code 01 – Read Coils

Function

This function serves to read individual output bits (coils) from the slave.

Start Address

The parameter **bit start address** is not checked by the driver and is sent unchanged.

Amount of Bits

Any value between **1** and **2008** is permitted as the **amount of bits** (number of coils).

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#1	Function Code
+2.0	bit_startadr	WORD	W#16#0040	Bit Start Address
+4.0	bit_count	INT	15	Amount of Bits

Example

Request Message FUNCTION 01:

05H	Slave Address
01H	Function Code
00H	Bit Start Address “High”
40H	Bit Start Address “Low”
00H	Amount of Bits “High”
0FH	Amount of Bits “Low”
xxH	LRC

Reply Message from Slave FUNCTION 01:

05H	Slave Address
01H	Function Code
02H	Byte Counter
01H	<Data> Coil 47H..40H
F7H	<Data> Coil 4EH..48H
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#7701	Data

The driver enters the data of the reply message into the destination DB **word-by-word.** The 1st received byte is entered as the Low Byte of the 1st word “data[1],” the 2nd received byte is entered as the High Byte of the 1st word “data[1],” and the 3rd received byte as the Low Byte of the 2nd word “data[2],” etc.

If an odd number of bytes are returned, the value **00H** is entered into the High Byte of the last word. Any unaccessed bits in the last received byte are masked to zero in the destination byte regardless of the received value.

5.2 Function Code 02 – Read Discrete Input

Function

This function serves to read individual input bits from the slave.

Start Address

The parameter **bit start address** is not checked by the driver and is sent unchanged.

Amount of Bits

Any value between **1** and **2008** is permitted as the **amount of bits** (number of DIs)

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#2	Function Code
+2.0	bit_startadr	WORD	W#16#0120	Bit Start Address
+4.0	bit_count	INT	24	Amount of Bits

Example

Request Message FUNCTION 02:

05H	Slave Address
02H	Function Code
01H	Bit Start Address “High”
20H	Bit Start Address “Low”
00H	Amount of Bits “High”
18H	Amount of Bits “Low”
xxH	LRC

Reply Message from Slave FUNCTION 02:

05H	Slave Address
02H	Function Code
03H	Byte Counter
04H	<Data> DI 127H..120H
26H	<Data> DI 12FH..128H
C8H	<Data> DI 137H..130H
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#2604	Data
+2.0	data[2]	WORD	W#16#00C8	Data

The driver enters the data of the reply message into the destination DB **word-by-word.** The 1st received byte is entered as the Low Byte of the 1st word “data[1],” the 2nd received byte is entered as the High Byte of the 1st word “data[1],” and the 3rd received byte as the Low Byte of the 2nd word “data[2],” etc.

If an odd number of bytes are returned, the value **00H** is entered into the High Byte of the last word. Any unaccessed bits in the last received byte are masked to zero in the destination byte regardless of the received value

5.3 Function Code 03 – Read Holding Registers

Function

This function serves to read individual registers from the slave.

Start Address

The parameter **Register start address** is not checked by the driver and is sent unchanged.

Amount of Register

A **maximum of 125 registers** (1 register = two bytes) can be read.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#3	Function Code
+2.0	reg_startadr	WORD	W#16#0040	Register Start Address
+4.0	reg_count	INT	2	Amount of Registers

Example

Request Message FUNCTION 03:

05H	Slave Address
03H	Function Code
00H	Register Start Address “High”
40H	Register Start Address “Low”
00H	Amount of Register “High”
02H	Amount of Register “Low”
xxH	LRC

Reply Message from Slave FUNCTION 03:

05H	Slave Address
03H	Function Code
04H	Byte Counter
21H	Register Address 40H Data “High”
23H	Register Address 40H Data “Low”
25H	Register Address 41H Data “High”
27H	Register Address 41H Data “Low”
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#2123	Data
+2.0	data[2]	WORD	W#16#2527	Data

5.4 Function Code 03 – Read 32-Bit Holding Registers

Function

This function serves to read individual 32-bit registers from the slave.

Start Address

The parameter **Register start address** is not checked by the driver and is sent unchanged.

Amount of Register

A **maximum of 62 registers** (1 register = four bytes) can be read.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#43	Function Code
+2.0	reg_startadr	WORD	W#16#0040	Register Start Address
+4.0	reg_count	INT	2	Amount of Registers

Request Message FUNCTION 03:

Example

05H	Slave Address
03H	Function Code
00H	Register Start Address “High”
40H	Register Start Address “Low”
00H	Amount of Register “High”
02H	Amount of Register “Low”
xxH	LRC

Reply Message from Slave FUNCTION 03:

05H	Slave Address
03H	Function Code
08H	Byte Counter
21H	Register Address 40H Data “Byte 1”
22H	Register Address 40H Data “Byte 2”
23H	Register Address 40H Data “Byte 3”
24H	Register Address 40H Data “Byte 4”
25H	Register Address 41H Data “Byte 1”
26H	Register Address 41H Data “Byte 2”
27H	Register Address 41H Data “Byte 3”
28H	Register Address 41H Data “Byte 4”
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	DWORD	W#16#21222324	Data
+4.0	data[2]	DWORD	W#16#25262728	Data

5.5 Function Code 04 – Read Input Registers

Function

This function serves to read individual registers from the slave.

Start Address

The parameter **Register start address** is not checked by the driver and is sent unchanged.

Amount of Register

A **maximum of 125 registers** (1 register = two bytes) can be read.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#4	Function Code
+2.0	reg_startadr	WORD	W#16#0050	Register Start Address
+4.0	reg_count	INT	3	Amount of Registers

Example

Request Message FUNCTION 04:

05H	Slave Address
04H	Function Code
00H	Register Start Address “High”
50H	Register Start Address “Low”
00H	Amount of Register “High”
03H	Amount of Register “Low”
xxH	LRC

Reply Message from Slave FUNCTION 04:

05H	Slave Address
04H	Function Code
04H	Byte Counter
31H	Register Address 50H Data “High”
32H	Register Address 50H Data “Low”
33H	Register Address 51H Data “High”
34H	Register Address 51H Data “Low”
35H	Register Address 52H Data “High”
36H	Register Address 52H Data “Low”
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#3132	Data
+2.0	data[2]	WORD	W#16#3334	Data
+4.0	data[3]	WORD	W#16#3536	Data

5.6 Function Code 05 – Write Single Coil

Function

This function serves to set or delete individual bits in the slave.

Bit Address

The parameter **Bit Address** is not checked by the driver and is sent unchanged.

Bit Status

The following two values are valid as the **Bit Status**:
FF00H → set bit to logical 1
0000H → reset bit to logical 0.

SEND Source DB

Structure of SEND Source Area:

Address	**Name**	**Type**	**Start value**	**Comment**
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#5	Function Code
+2.0	bit_address	WORD	W#16#0019	Bit Address
+4.0	bit_state	WORD	W#16#FF00	Bit Status

Example

Request Message FUNCTION 05:

05H	Slave Address
05H	Function Code
00H	Bit Address “High”
19H	Bit Address “Low”
FFH	Set Bit
00H	
xxH	LRC

Reply Message from Slave FUNCTION 05:

05H	Slave Address
05H	Function Code
00H	Bit Address “High”
19H	Bit Address “Low”
FFH	Bit Status “High”
00H	Bit Status “Low”
xxH	LRC

5.7 Function Code 06 – Write Single Register

Function

This command serves to overwrite a slave register with a new value.

Register Address

The parameter **Register Address** is not checked by the driver and is sent unchanged.

Register Value

Any value can be used as the **Register Value**

SEND Source DB

Structure of SEND Source Area:

Address	**Name**	**Type**	**Start value**	**Comment**
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#6	Function Code
+2.0	Reg_address	WORD	W#16#0180	Register Address
+4.0	Reg_count	WORD	W#16#3E7F	Registers Value

Example

Request Message FUNCTION 06:

05H	Slave Address
06H	Function Code
01H	Register Address “High”
80H	Register Address “Low”
3EH	Register Value “High”
7FH	Register Value “Low”
xxH	LRC

Reply Message from Slave FUNCTION 06:

05H	Slave Address
06H	Function Code
01H	Register Address “High”
80H	Register Address “Low”
3EH	Register Value “High”
7FH	Register Value “Low”
xxH	LRC

5.8 Function Code 06 – Write Single 32-Bit Register

Function

This command serves to overwrite a slave 32-bit register with a new value.

Register Address

The parameter **Register Address** is not checked by the driver and is sent unchanged.

Register Value

Any value can be used as the **Register Value**
Structure of SEND Source Area:

SEND Source DB

Address	**Name**	**Type**	**Start value**	**Comment**
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#46	Function Code
+2.0	reg_address	WORD	W#16#0180	Register Address
+4.0	reg_count	DWORD	W#16#11223344	Registers Value

Example

Request Message FUNCTION 06:

05H	Slave Address
06H	Function Code
01H	Register Address “High”
80H	Register Address “Low”
11H	Register Value “Byte 1”
22H	Register Value “Byte 2”
33H	Register Value “Byte 3”
44H	Register Value “Byte 4”
xxH	LRC

Reply Message from Slave FUNCTION 06:

05H	Slave Address
06H	Function Code
01H	Register Address “High”
80H	Register Address “Low”
11H	Register Value “Byte 1”
22H	Register Value “Byte 2”
33H	Register Value “Byte 3”
44H	Register Value “Byte 4”
xxH	LRC

5.9 Function Code 07 - Read Exception Status

Function

This function code serves to read 8 event bits of the connected slave.

The start bit number of the event bit is determined by the connected device. Therefore it has not to be specified by the SIMATIC user program.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#7	Function Code

Example

Request Message FUNCTION 07:

05H	Slave Address
07H	Function Code
xxH	LRC

Reply Message from Slave FUNCTION 07:

05H	Slave Address
07H	Function Code
3EH	<Data>
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#3Exx	Data

The driver enters the individual bits of the reply message into the **High Byte i**n the destination DB data[1]. The Low Byte of data[1] remains unchanged. Value 1 is displayed as the length in parameter LEN of P_RCV_RK.

5.10 Function Code 08 – Diagnostics (Loop Back Test)

Function

This function serves to check the communications connection. Only **Diagnostic Code 0000** is supported with this function code.

Diagnostic Code

The only permissible value for the parameter Diagnostic Code is 0000.

Test Value

Any value can be used as the **Test Value**.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#8	Function Code
+2.0	diag_code	WORD	W#16#0000	Diagnostic Code
+4.0	test_value	WORD	W#16#A5C3	Test Value

Example

Request Message FUNCTION 08:

05H	Slave Address
08H	Function Code
00H	Diagnostic Code “High”
00H	Diagnostic Code “Low”
A5H	Test Value “High”
C3H	Test Value “Low”
xxH	LRC

Reply Message from Slave FUNCTION 08:

05H	Slave Address
08H	Function Code
00H	Diagnostic Code “High”
00H	Diagnostic Code “Low”
A5H	Test Value “High”
C3H	Test Value “Low”
xxH	LRC

The slave must return the request message to the master unchanged (echo). The reply message is not entered into an RCV DB.

5.11 Function Code 11 – Get Comm Event Counter

Function

This function code serves to read a **“Status Word”** (2 bytes long) and an **“Event Counter”** (2 bytes long) from the slave.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	Address	BYTE	B#16#5	Slave Address
+1.0	Function	BYTE	B#16#0B	Function Code

Example

Request Message FUNCTION 11:

05H	Slave Address
0BH	Function Code
xxH	LRC

Reply Message from Slave FUNCTION 11:

05H	Slave Address
0BH	Function Code
FEH	Status Word “High”
DCH	Status Word “Low”
01H	Event Counter “High”
08H	Event Counter “Low”
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#FEDC	Status Word
+2.0	data[2]	WORD	W#16#0108	Event Counter

5.12 Function Code 12– Get Comm Event Log

Function

This function code serves to read a:

- 2 Byte **“Status Word”**
- 2 Byte **“Event Counter”**
- 2 Byte **“Message Counter”** and
- 64 Byte **“Event Bytes”**

from the slave.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#0C	Function Code

Example

Request Message FUNCTION 12:

05H	Slave Address
0CH	Function Code
xxH	LRC

Reply Message from Slave FUNCTION 12:

05H	Slave Address
0CH	Function Code
46H	Byte Counter
87H	Status Word “High”
65H	Status Word “Low”
01H	Event Counter “High”
08H	Event Counter “Low”
02H	Message Counter “High”
20H	Message Counter “Low”
01H	Event Byte 1
12H	Event Byte 2
C2H	Event Byte 63
D2H	Event Byte 64
xxH	LRC

RCV Destination DB

Contents of RCV Destination Area:

Address	Name	Type	Actual value	Comment
+0.0	data[1]	WORD	W#16#8765	Status Word
+2.0	data[2]	WORD	W#16#0108	Event Counter
+4.0	data[3]	WORD	W#16#0220	Message Counter
+6.0	bytedata[1]	BYTE	B#16#01	Event Byte 1
+7.0	bytedata[2]	BYTE	B#16#02	Event Byte 2
:	:			:
+68.0	bytedata[63]	BYTE	B#16#C2	Event Byte 63
+68.0	bytedata[64]	BYTE	B#16#C3	Event Byte 64

5.13 Function Code 15 – Write Multiple Coils

Function

This function code serves to change up to 1976 bits in the slave.

Start Address

The parameter **Bit Start Address** is not checked by the driver and is sent unchanged.

Amount of Bits

Any value between **1** and **1976** is permitted as the **amount of bits** (number of coils). This indicates how many bits in the slave should be overwritten.

The parameter “byte counter” in the request message is generated by the driver based on the transferred parameter “amount of bits.” It is not included in the SEND Source DB.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	Address	BYTE	B#16#5	Slave Address
+1.0	Function	BYTE	B#16#0F	Function Code
+2.0	bit_startadr	WORD	W#16#0050	Bit Start Address
+4.0	bit_count	INT	10	Amount of Bits
+6.0	coil_state[1]	WORD	W#16# EFCD	Status Coil 5FH..58H/57H..50H

Example

Request Message FUNCTION 15:

05H	Slave Address
0FH	Function Code
00H	Bit Address “High”
50H	Bit Address “Low”
00H	Amount of Bits “High”
0AH	Amount of Bits” Low”
02H	Byte Counter
CDH	Status Coil 57H ..50H
EFH	Status Coil 59H ..58H
xxH	LRC

Reply Message from Slave FUNCTION 15:

05H	Slave Address
0FH	Function Code
00H	Bit Address “High”
50H	Bit Address “Low”
00H	Amount of Bits “High”
0AH	Amount of Bits “Low”
xxH	LRC

5.14 Function Code 16 – Write Multiple Registers

Function

Function code 16 serves to overwrite up to **123 registers** in the slave with one request message.

Start Address

The parameter **Register Start Address** is not checked by the driver and is sent unchanged.

Amount of Registers

A **maximum of 123 registers** (1 register = two bytes) can be written.

The parameter “byte counter” in the request message is generated by the driver based on the transferred parameter “amount of registers.” It is not included in the SEND Source DB.

SEND Source DB

Structure of SEND Source Area:

Address	**Name**	**Type**	**Start value**	**Comment**
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#10	Function Code
+2.0	reg_startadr	WORD	W#16#0060	Register Start Address
+4.0	reg_count	INT	3	Amount of Registers
+6.0	reg_data[1]	WORD	w#16#41A1	Register Data
+8.0	reg_data[2]	WORD	w#16#42A2	Register Data
+10.0	reg_data[3]	WORD	w#16#43A3	Register Data

Example

Request Message FUNCTION 16:

05H	Slave Address
10H	Function Code
00H	Register Address “High”
60H	Register Address “Low”
00H	Amount of Registers “High”
03H	Amount of Registers” Low”
06H	Byte Counter
41H	<reg_data[1]> “High”
A1H	<reg_data[1]> “Low”
42H	<reg_data[2]> “High”
A2H	<reg_data[2]> “Low”
43H	<reg_data[3]> “High”
A3H	<reg_data[3]> “Low”
xxH	LRC

Reply Message from Slave FUNCTION 16:

05H	Slave Address
10H	Function Code
00H	Register Address “High”
60H	Register Address “Low”
00H	Amount of Registers “High”
03H	Amount of Registers “Low”
xxH	LRC

5.15 Function Code 16 – Write Multiple 32-Bit Registers

Function

Function code 16 serves to overwrite up to **61 registers** in the slave with one request message.

Start Address

The parameter **Register Start Address** is not checked by the driver and is sent unchanged.

Amount of Registers

A **maximum of 61 registers** (1 register = four bytes) can be written.
The parameter “byte counter” in the request message is generated by the driver based on the transferred parameter “amount of registers.” It is not included in the SEND Source DB.

SEND Source DB

Structure of SEND Source Area:

Address	Name	Type	Start value	Comment
+0.0	address	BYTE	B#16#5	Slave Address
+1.0	function	BYTE	B#16#50	Function Code
+2.0	reg_startadr	WORD	W#16#0120	Register Start Address
+4.0	reg_count	INT	2	Amount of Registers
+6.0	reg_data[1]	DWORD	w#16#51A152A2	Register Data
+10.0	reg_data[2]	DWORD	w#16#53A354A4	Register Data

Example

Request Message FUNCTION 16:

05H	Slave Address
10H	Function Code
01H	Register Address “High”
20H	Register Address “Low”
00H	Amount of Registers “High”
02H	Amount of Registers” Low”
08H	Byte Counter
51H	<reg_data[1]> “Byte 1”
A1H	<reg_data[1]> “Byte 2”
52H	<reg_data[1]> “Byte 3”
A2H	<reg_data[1]> “Byte 4”
53H	<reg_data[2]> “Byte 1”
A3H	<reg_data[2]> “Byte 2”
54H	<reg_data[2]> “Byte 3”
A4H	<reg_data[2]> “Byte 4”
xxH	LRC

Reply Message from Slave FUNCTION 16:

05H	Slave Address
10H	Function Code
01H	Register Address “High”
20H	Register Address “Low”
00H	Amount of Registers “High”
02H	Amount of Registers “Low”
xxH	LRC

6 CPU – CP Interface

Used SFBs

Data transfer between CP and CPU is carried out by means of FBs **P_SND_RK (FB8)** and **P_RCV_RK (FB7)**.

FB P_SND_RK is activated by an edge at input **REQ**, when data output is required. FB P_RCV_RK is made ready to receive by **EN_R=1**.

A P_RCV_RK is required for all reading function codes.

Parallel Processing of Requests

At a given time, only one FB P_SND_RK and one FB P_RCV_RK can be called for each CP 341 in the user program.

6.1.1 Data Transfer from CPU to CP with P_SND_RK

Activation

Execution of a Modbus function code is activated by means of an SFB **P_SND_RK** with an **edge** at input **REQ**.

Enter ‘S’ for SEND at the SF parameter. The logical module address is entered at LADDR. You must enter ‘X’ for expanded data block as the area type of the partner CPU. No values must be specified for the other parameters of the partner CPU (R_...).

This ensures transfer to the driver of the function codes required for the execution.

Data Source

When P_SND_RK is activated, the **source data area** specified with the parameters **DB_NO** and **DBB_NO** is transferred to the CP with the length **LEN.**

Length Indication

The length **LEN** depends on the function code used.

Function Code	*Length LEN in Byte*
01	*6*
02	*6*
03	*6*
04	*6*
05	*6*
06 *16-Bit Register*	*6*
06 *32-Bit Register*	*8*
07	*2*
08	*6*
11	*2*
12	*2*
15	*>=8*
16 *16-Bit Register*	*>=8*
16 *32-Bit Register*	*>=10*

If the transferred data quantities differ from those listed above for the individual function codes, the job is not carried out and P_SND_RK rejects it with an edge at output ERROR.

The data length LEN may exceed the required amount of data for the activated function. The driver checks the data length according to function code and amount of bits/registers. If less data than necessary are transferred to the CP, the send job is finished with error.
So it is not necessary to calculate LEN for each send job, when the maximum length is used. But it takes some more time for data transfer CPU → CP because more data than needed are transferred.

SEND Source DB

The parameters required for the execution of a function code must be entered as user data in the source data area. A detailed description of each P_SND_RK source DB can be found in the chapter “Function Codes.”

Generation of Messages

The request messages to the slave are generated in accordance with the transferred P_SND_RK source data and sent by the CP.

First of all the driver checks if the length LEN specified at P_SND_RK corresponds to the length for this function code. If it does not, the job is not carried out and it is completed with an edge at output ERROR of the P_SND_RK.

When using function codes other than those listed, the activated job is not carried out either and is completed with ERROR at P_SND_RK.

The elements “byte counter” and “LRC” in the request message are generated by the CP; an entry in the P_SND_RK source DB is not required.

Job Completion for Writing Functions

For writing function codes, the activated P_SND_RK is completed after a reply message is received without error. This is communicated to the SIMATIC user program by means of an edge at output **DONE** of the P_SND_RK.

If **errors** were recognized during the message exchange, or if the slave sends an **error code** reply message, this is reported by an edge at output **ERROR**.

Job Completion for Reading Functions

For reading functions, the activated P_SND_RK is completed after the reply message is received without error **and** complete transfer of the received data to the CPU. This is communicated to the SIMATIC user program by means of an edge at output **DONE** of the P_SND_RK.

This means that the received data are already available on the CPU.

If **errors** were recognized during the message exchange, or if the slave sends an **error code** reply message, this is reported by an edge at output **ERROR**. In this case no receive data are transferred to the CPU.

STATUS Entry on Job Completion

For those instances when a job is completed with **ERROR** at P_SND_RK, an additional error code is entered in the **STATUS** parameter. The exact cause for the error can be determined with this error code.

6.1.2 Data Transfer from CPU to CP with P_RCV_RK

Data Destination

All **reading** function codes require a P_RCV_RK. When FB P_RCV_RK is ready to receive, it accepts the received data from the CP and enters them into the data destination specified in the parameters **DB_N0** and **DBB_N0**.

How Data Reception is Displayed

The user is informed of data reception in the CPU by means of an edge at output **NDR**.

At this point the length of the received data block is displayed in the parameter **LEN**.

As mentioned above, completion of the entire Modbus job can be recognized at output DONE of FB P_SND_RK.

How to Handle an Error

In the event of receive or transfer errors, there is no data transfer to the CPU.
In this instance P_SND_RK is completed with an edge at the output ERROR.

P_RCV_RK Destination DB

The user data received with a reading function code are entered into the P_RCV_RK destination area.

A detailed description of each P_RCV_RK destination DB can be found in the chapter “Function Codes.”

The length of data entered is displayed on the parameter LEN of P_RCV_RK.

7 Diagnostics of the Driver

Diagnostics Functions

The diagnostics functions of the CP enable you to easily know when an error has occurred and quickly determine the cause of the problem. The following diagnostic facilities are available:

- Diagnostics via display elements of the CP
- Diagnostics via the STATUS output of the function blocks
- Diagnostic buffer of the CP

Display Elements (LED)

The display elements provide information on the operating status and/or a possible error status of the CP. The display elements give a first overview of internal or external errors, as well as interface-specific errors.

STATUS Output of FBs

Each function block has a STATUS output for error diagnostics purposes. Reading this STATUS output enables the user to obtain information on errors which occurred during communication. The STATUS parameter can be evaluated in the user program.

Diagnostic Buffer of the CP

All errors / events described in Section 7.3 are also entered in the diagnostic buffer of the CP. The manual for the CP describes how you can read the diagnostic buffer.

7.1 Diagnostics via Display Elements (LEDs)

Introduction

The display LEDs of the CP 341 provide general operational information. The following different display functions are available:

- **Group Error Displays**

- SF (red) Error occurred or new parameters assigned

- **Special Displays**

- TXD (green)	Send active; lights up when the CP 341 sends user data via the interface
- RXD (green)	Receive active; lights up when the CP 341 receives user data via the interface

Group Error Display SF

The group error display SF always lights up after power-on and goes out after initialization is complete. If parameter assignment data were created for the CP 341, the SF LED lights up again briefly when new parameters are loaded.

The group error display SF lights up, when the following errors have occurred:

- Hardware error
- Firmware error
- Parameter assignment error
- BREAK (Receiving line between CP 341 and communication partner is interrupted or CTS or DSR signals not asserted at the connector.)

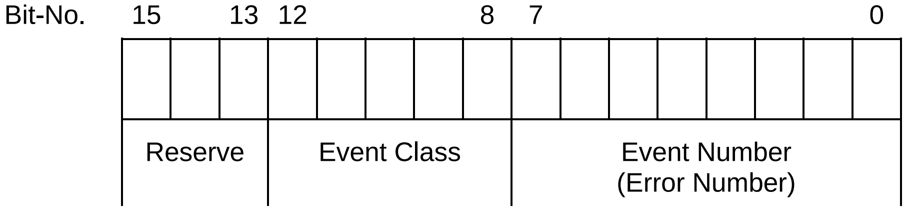
7.2 Diagnostic Messages of the Function Blocks

Introduction

Each function block has a STATUS parameter for error diagnostics purposes. Each STATUS message number has the same meaning, independent of the system function block used.

Event Class / Event Number Numbering Scheme

The following figure shows the structure of the STATUS parameter.



The individual errors / events are listed in Section 7.3

7.3 Table of Errors / Events

Event Classes

The following event classes are defined:

Event Class	Description	Described in
1	Hardware error on CP	CP Manual
2	Error during initialization	CP Manual
3	Error during parameter assignment of PBK	CP Manual
4	Errors in CP – CPU data traffic recognized by CP	CP Manual
5	Error during processing of a CPU job	CP Manual, Driver Manual
6	Error during processing of a partner job	CP Manual
7	Send error	CP Manual
8	Receive error	Driver Manual
9	Error code message received from link partner	Not used
10	Errors recognized by CP in reaction message from partner	Not used
14	General processing errors of the loadable driver	Driver Manual

7.3.1 Error Codes for “CPU Job Errors”

Event Class 5 (05H) **“CPU Job Errors“**			
Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
05 18H	24	Transmission length during transmission is too large (> 4 Kbytes), or transmission length for SEND is too small.	Check the parameter LEN for SEND.

7.3.2 Error Codes for “Receive Errors”

Event Class 8 (08$_H$) **“CPU Receive Errors“**			
Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
08 06$_H$	6	Character delay time exceeded.	Eliminate error in partner device or interference on the transmission line or increase the value of the “Character Delay Time” parameter.
08 0C$_H$	6	Transmission error (parity error, overflow error, stop bit error (frame)) recognized in a character.	Check for interference which could influence the transmission line. If required, change system structure and/or cable laying. Check whether the protocol parameters transmission rate, amount of data bits, parity, and amount of stop bits have the same settings for the CP and the link partner.
08 0D$_H$	6	BREAK Receiving line to partner device is interrupted.	Establish connection between the devices or switch on partner device. Make sure CTS and DSR are asserted at the CP connector. For use with TTY operation, check line current at idle state. For use with an RS422/485 (X27) connection, check and, if required, change the connector pin assignment of the 2-wire receiving line R(A), R(B).
08 18$_H$	24	DSR = OFF or CTS = OFF	The partner has switched the DSR or CTS signal to ”OFF” before or during a transmission. Check the partner’s control of the RS 232C secondary signals.

Event Class 8 (08H) **“CPU Receive Errors“**			
08 30H	48	A request message has been sent and the reply monitoring time has elapsed without the start of a reply message being recognized.	Check if transmission line is interrupted (interface analyzer may be required). Check if the protocol parameters transmission rate, amount of data bits, parity, and amount of stop bits have the same settings in CP and the link partner. Check if the value for the “Response Time-out” parameter is big enough. Check if the specified slave address exists.
08 32H	50	Overflow of receive buffer in CP during reception of the reply message.	Check protocol settings for the slave.
08 33H	51	A wrong start character was received. The start character was not a colon (3AH).	Check protocol settings for the slave.
08 34H	52	A start character was received within a telegram. The first part of the telegram is discarded and reception starts again with the second start character.	Check if transmission line is interrupted (interface analyzer may be required). This does not in itself fail the send job. The error only appears in the CP diagnostic buffer.

7.3.3 Error Codes for “General Processing Errors”

Event Class E (0EH) **“General Processing Errors“**			
Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
0E 01H	1	Error during initialization of the driver-specific SCC process	Reassign parameters of driver and reload.
0E 02H	2	Error during initialization of the driver-specific SCC process	Reassign parameters of driver and reload.
0E 03H	3	Error during startup of driver: Wrong data transfer process active (interface to SFBs). The driver cannot function with this data transfer process.	Reassign parameters of driver and reload.
0E 04H	4	Error during startup of driver: Illegal interface submodule. The driver cannot run with the parameterized interface submodule.	Check and correct parameter assignment.
0E 05H	5	Error with driver dongle: No dongle plugged in, or inserted dongle is faulty. The driver is not ready to run.	Check if a driver dongle is plugged into the CP. If the inserted dongle is faulty, replace it with a correct dongle.
0E 06H	6	Error with driver dongle: The dongle has no valid contents. The driver is not ready to run.	Obtain a correct dongle from the Siemens office which supplied you with the driver.
0E 10H	16	Internal error procedure: default branch in procedure automatic device.	Restart CP (Mains_ON)
0E 11H	17	Internal error procedure: default branch for procedure status Send / Receive.	Restart CP (Mains_ON)
0E 12H	18	Internal error active automatic device: default branch.	Restart CP (Mains_ON)
0E 13H	19	Internal error passive automatic device: default branch.	Restart CP (Mains_ON)

Event Class E (0EH) “Loadable Driver – General Processing Errors <Parameter Assignment>“			
Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
0E 20H	32	For this data link, the amount of data bits must be set to 7. The driver is not ready to run.	Correct parameter assignment of the driver. Load driver parameters.
0E 21H	33	The Character Delay Time parameter is not within the range of 1 to 6500 milliseconds. The driver is operating with a default value of 1000 milliseconds	Correct parameter assignment of the driver. Load driver parameters
0E 22H	34	The operating mode set for the driver is illegal. “Normal operation” or “Interference Suppression” must be specified. The driver is not ready to run.	Correct parameter assignment of the driver. Load driver parameters.
0E 23H	35	An illegal value for parameter Response Time-out has been set: Valid values are 5 to 65500ms. The driver is operating with a default value of 2000 milliseconds.	Correct parameter assignment of the driver. Load driver parameters.
0E 2EH	46	An error occurred when reading the interface parameter file. The driver is not ready to run.	Restart CP (Mains_ON)

Event Class 5 (05H) “Loadable Driver – General Processing Errors <CPU - CP>“			
Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
0E 30H	48	Internal error during data transfer to CPU: Unexpected acknowledgment Passive.	Can be ignored if it happens intermittently.
0E 31H	49	Timeout during data transfer to CPU.	Check CP-CPU interface.
0E 32H	50	Error occurred during data transfer to CPU with RCV: Exact failure reason (detailed error) is in diagnostic buffer before this entry.	Check CP-CPU interface.
0E 33H	51	Internal error during data transfer to CPU: Illegal status of automatic device.	Check CP-CPU interface.

Event Class 5 (05H) **“Loadable Driver – General Processing Errors <Processing of a Send Job>“**			
Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
0E 40H	64	Value specified for parameter LEN at SFB SEND too small.	Minimum length is 2 bytes.
0E 41H	65	Value specified for parameter LEN at SFB SEND too small. A greater length is required for the transferred function code.	The minimum length for this function code is 6 bytes.
0E 42H	66	Transferred function code is illegal.	The only function codes which are permitted are those listed in the chapter “Function Codes.”
0E 43H	67	Slave Address 0 (= Broadcast) not permitted with this function code.	Only use Slave Address 0 for the suitable function codes.
0E 44H	68	The value of the transferred parameter “Amount of Bits” is not within the range 1 to 2008.	Correct your source DB
0E 45H	69	The value of the transferred parameter “Amount of Registers” is not within the range 1 to 125 or, for 32-bit registers, 1 to 62.	Correct your source DB
0E 46H	70	Function codes 15 or 16: The values of the transferred parameters “Amount of Bits” and/or “Amount of Registers” are not within the range 1 to 1976 and/or 1 to 123 and/or, for 32-bit registers, 1 to 61.	Correct your source DB.
0E 47H	71	Function codes 15 or 16: The parameter LEN for SFB SEND does not correspond to the transferred parameters “Amount of Bits” and/or “Amount of Registers.” Parameter LEN is too small.	Increase parameter LEN for SEND until a sufficient amount of user data is transferred to the CP. A larger amount of user data must be transferred to the CP because of the “Amount of Bits” and/or “Amount of Registers.”
0E 48H	72	Function code 5: The code specified in SEND source DB for “Set Bit” (FF00H) or “Delete Bit” (0000H) is wrong.	Correct your source DB. Only the value FF00H or 0000H are allowed for writing a coil.
0E 49H	73	Function code 8: The code specified in SEND source DB for “Diagnostic Code” is wrong.	The only permitted code is “Diagnostic Code” 0000H.
0E 4AH	74	Access to 32-bit registers is only allowed with function code 03, 06, 16. (Bit 2^6 of function code in source DB is set.)	Correct your source DB.
0E 4FH	79	The R_TYP specified for SFB SEND RK is illegal with this driver.	‘X’ has to be entered for R_TYP in P_SND_RK.

Event Class E (0EH)
“Loadable Driver – General Processing Errors <Receive Evaluation>“

Event Class/ No. (Hex)	**Event Number (Decimal)**	**Event Text**	**Remedy**
0E 50H	80	Slave address incorrect: The received slave address is different from the sent slave address.	The wrong slave has replied. Check if the transmission line is interrupted (interface analyzer may be required).
0E 51H	81	Function code incorrect: The function code received in the reply message is different from the sent function code.	Check slave device.
0E 52H	82	Byte Underflow: Amount of characters received is less than should have resulted from the byte counter of the reply message, or is less than expected with this function code.	Check slave device. If working with 32-bit registers, check accessed address of the slave whether it belongs to the expected memory area (16/32-bit).
0E 53H	83	Byte Overflow: Amount of characters received is more than should have resulted from the byte counter of the reply message, or is more than expected with this function code.	Check slave device. If working with 32-bit registers, check accessed address of the slave whether it belongs to the expected memory area (16/32-bit).
0E 54H	84	Byte counter wrong: The byte counter received in the reply message is too small.	Check slave device.
0E 55H	85	Byte counter wrong: The byte counter received in the reply message is wrong.	Check slave device.
0E 56H	86	Echo wrong: The data of the reply message (amount of bits, ...) echoed from the slave are different from the data sent in the request message.	Check slave device.
0E 57H	87	LRC incorrect: An error has occurred on checking the LRC of the reply message from the slave.	Check slave device.
0E 58H	88	A received character within the reply message is not an ASCII character (0-9, A-F)	Check slave device. Make sure it is in ASCII mode and not RTU.

Event Class 5 (05H) **“CPU Job Errors“**			
Event Class/ No. (Hex)	Event Number (Decimal)	Event Text	Remedy
0E 61H	97	Reply message with Modbus Exception Code 01: Illegal Function	See manual for slave device or Modbus Protocol Specification
0E 62H	98	Reply message with Modbus Exception Code 02: Illegal Data Address	See manual for slave device or Modbus Protocol Specification
0E 63H	99	Reply message with Modbus Exception Code 03: Illegal Data Value	See manual for slave device or Modbus Protocol Specification
0E 64H	100	Reply message with Modbus Exception Code 04: Failure in associated device	See manual for slave device or Modbus Protocol Specification
0E 65H	101	Reply message with Modbus Exception Code 05: Acknowledge	See manual for slave device or Modbus Protocol Specification
0E 66H	102	Reply message with Modbus Exception Code 06: Busy, Rejected message	See manual for slave device or Modbus Protocol Specification
0E 67H	103	Reply message with Modbus Exception Code 07: Negative Acknowledgment	See manual for slave device or Modbus Protocol Specification

8 Application Example

General Information

The following simple programming example illustrates the use of FBs P_SND_RK and P_RCV_RK.

When the Modbus master is installed, the application example is stored in the STEP 7 directory EXAMPLES under the name *MB_ASCII*.

The S7 program is for information purposes only and is not to be understood as a solution for a customer-specific installation configuration.

In order to illustrate the basic structure, we intentionally kept it simple and avoided symbolic display.

8.1 Used Blocks

Used Blocks

The following blocks are used in the programming example.

Block	Symbol	Comment
OB 1	Cycle Execution	Cyclic program processing
OB 100	Complete Restart	Startup OB for Restart
FC 10	Initiation	FC for Startup OB
FC 21	Execute Send Jobs	FC Calling P_SND_RK
FC 22	Execute Receive Jobs	FC Calling FB P_RCV_RK
DB50	IDB_P_SND_RK	Instance DB for P_SND_RK
DB70	I_DB_P_RCV_RK	Instance DB for P_RCV_RK
DB40	Work DB Send	Work DB for FC21 and P_SND_RK
DB41	Work DB Receive	Work DB for FC23 and P_RCV_RK
DB42	SOURCE_DB	P_SND_RK Source DB with send data
DB43	DESTINATION_DB	P_RCV_RK Destination DB for receive data

Used Data

The following operands (memory bits, data bits, or data words) are used in the programming example.

Operand	Symbol	Comment
M 120.7		Trigger bit for the execution of a P_SND_RK job
DB40.DBX.0.0		Control parameter REQuest: for activating a P_SND_RK
DB40.DBX.0.1		Control parameter Reset: for aborting current P_SND_RK
DB40.DBX.0.4		Status parameter DONE: Indicates that current P_SND_RK was completed without error
DB40.DBX.0.5		Status parameter ERROR: Indicates that current P_SND_RK was completed with error
DB40.DBW.2		Success counter for P_SND_RK
DB40.DBW.6		Success counter for P_SND_RK
DB40.DBW.8		Error counter for P_SND_RK
DB40.DBW.10		Length LEN of P_SND_RK source data area to be transferred to the CP in bytes
DB40.DBW.12		STATUS display in P_SND_RK
DB40.DBW.14		Stored P_SND_RK STATUS display
DB41.DBX.0.0		Control parameter EN_R: P_RCV_RK ready to receive
DB41.DBX.0.4		Status parameter NDR: Indicates that current P_RCV_RK has received new data from the CP
DB41.DBX.0.5		Status parameter ERROR: Indicates that current P_RCV_RK has been completed with error
DB41.DBW.4		Stored length LEN of P_RCV_RK
DB41.DBW.6		Success counter for P_RCV_RK
DB41.DBW.8		Error counter for P_RCV_RK
DB41.DBW.10		Length LEN of P_RCV_RK destination data area received by the CP in bytes
DB41.DBW.12		STATUS display in P_RCV_RK
DB41.DBW.14		Stored P_RCV_RK STATUS display

8.1.1 Program Description

General Information

The programming example consists of:

- Startup block OB100, FC10
- Cyclic part OB1 calling
- Function block FC21 for data transfer CPU to CP (**Send**)
- FC23 to **receive** data CP to CPU

The parameters for the programmed system function blocks P_SND_RK, P_RCV_RK are stored in the work DBs **DB40** and **DB41**.

The send data (SEND source area) are contained in **DB42**.
Data received from the link partner are entered into **DB43** (RCV destination area).

P_SND_RK Job

A **P_SND_RK job** can be activated in the cyclic part of the program by setting memory bit **M 120.7** (for example, by CONTROL VARIABLE).
The data with length LEN contained in the P_SND_RK source area DB42 are transferred to the CP.
Memory bit M 120.7 is reset immediately.

After completion of the P_SND_RK job without error, a success counter is incremented; after completion with error, an error counter is incremented.

P_RCV_RK Job

An SFB **P_RCV_RK** is programmed in FC23, where the **Receive Enable** is always **"1,"** in order to receive data from the link partner.
The receive data are entered into the **P_RCV_RK destination area**, the amount of entered data is displayed in parameter **LEN.**

On taking on data without error, the success counter is incremented; on completion with error, the error counter is incremented.

For the P_SND_RK and P_RCV_RK jobs, the output parameter **STATUS** is stored when a value other than 0 is reported.

8.1.2 Programming Example

Programming Example

The blocks are listed as follows:

Block	Comment
OB 100	Startup OB for Restart
FC 10	FC for Startup OB
OB 1	Cyclic program processing
FC 21	FC Calling P_SND_RK
FC 22	FC Calling FB P_RCV_RK

Program Startup

```
OB100           Start-Up-OB

        L       272                             //logical address
        T       DB40.DBW     16                 //for SEND
        T       DB41.DBW     16                 /and RCV
        UC      FC      10                      //Call of FC for Initiation
```

```
FC10            Initiation

//------------------------------------------------
        Reset Control Bits
//------------------------------------------------
        L       B#16#0
        T       DB40.DBB     0                  //SEND- Work-DB
        T       DB41.DBB     0                  //RCV- Work-DB
//------------------------------------------------
        Reset counters/STATUS
//------------------------------------------------
        L       W#16#0
        T       DB40DBW      6                  //SEND- Work-DB
        T       DB40DBW      8
        T       DB40DBW      12
        T       DB40DBW      14
        T       DB41DBW      6                  //RCV- Work-DB
        T       DB41DBW      8
        T       DB41DBW      12
        T       DB41DBW      14
```

Cyclic Program Sequence

```
OB1             Cyclic-OB
```

```
        UC      FC      21                      //Call of SEND
        UC      FC      23                      //Call of RCV
```

```
FC21            Execute SEND-Jobs
```

```
//      ------------------------------------------------
//      Interlockings for SEND
//      ------------------------------------------------
        U       M               120.7           //Trigger SEND
        U       N       DB40.DBX        0.0     //SEND_REQ
        U       N       DB40.DBX        0.4     //SEND_DONE
        U       N       DB40.DBX        0.5     //SEND_ERROR
        R       M               120.7           //Reset Trigger SEND
        S       DB40.DBX                0.0     //Set SEND_REQ
//      ------------------------
//      Generate edge SEND_REQ
//      ------------------------
        U(
        O       DB40.DBX                0.4     //SEND_DONE
        O       DB40.DBX                0.5     //SEND_ERROR
        )
        U       DB40.DBX                0.0     //SEND_REQ
        R       DB40.DBX                0.0     //SEND with REQ=0
//      ------------------------
//      Supply LEN
//      ------------------------
        L       W#16#20                         //Length SEND-Data
        T       DB40.DBW                10      //SEND-LEN
//      ------------------------
//      SEND with Instance-DB
//      ------------------------
        CALL    FB      8 ,     DB50
        SF:=
        REQ     :=DB40.DBX0.0
        R       :=DB40.DBX0.1
        LADDR  :=DB40.DBW16
        DB_NO  :=42
        DBB_NO:=10
        LEN      :=DB40.DBW10
        R_CPU_NO:=
        R_TYP:='x'
        R_NO:=
        R_OFFSET:=
        R_CF_BYT:=
        R_CF_BIT:=
        DONE    :=DB40.DBX0.4
        ERROR  :=DB40.DBX0.5
        STATUS:=DB40.DBW12
```

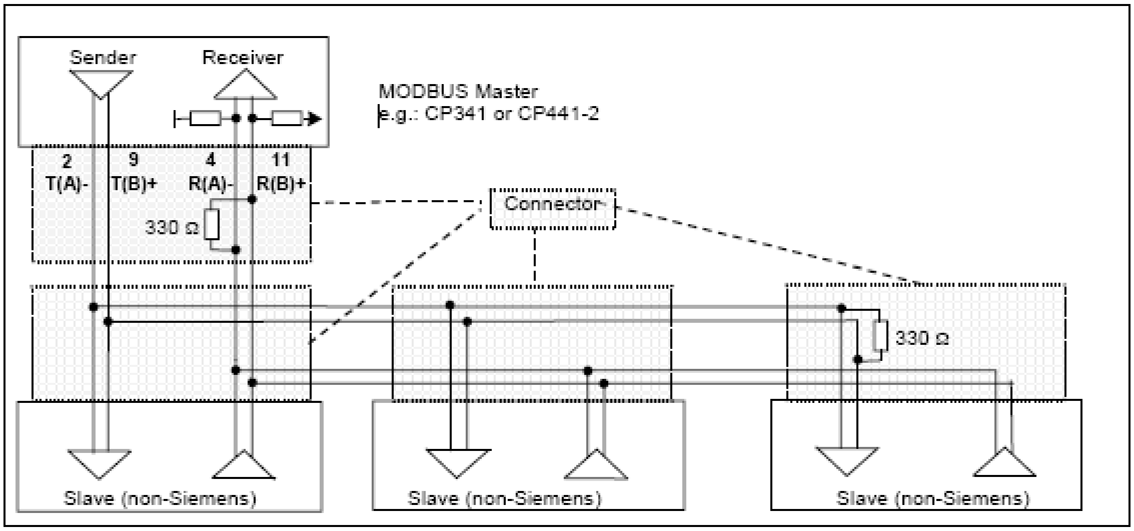
```
//        -------------------------------------------
//        Check “Complete without error“
//        -------------------------------------------
          U         DB40.DBX        0.4                 //DONE ?
          SPBN      CON1                                //if NO
          L         DB40.DBW        6                   //“Complete without
                                                        //error“
          +1                                            //increment counter
          T         DB40.DBW        6
          :                                             // :
          :                                             //further user
          :                                             // functions
          :                                             // :
          SPA       LEAV
//        -------------------------------------------
//        Check “Complete with error“
//        -------------------------------------------
CON1:     U         DB40.DBX        0.5                 //ERROR ?
          SPBN      CON2                                //if NO
          L         DB40.DBW        8                   //“Complete with error“
          +1                                            // increment counter
          T         DB40.DBW        8
          :                                             // :
          :                                             //Error-Handling
          :                                             // :
          L         0
          L         DB40.DBW        12                  //if STATUS <>0
          ==I
          SPB       LEAV
          T         DB40.DBW        14                  //save STATUS
          SPA       LEAV
//        -------------------------------------------
//        Check “Error in STATUS“
//        -------------------------------------------
CON2:     L         0
          L          DB40.DBW       12                  //if STATUS <>0
          ==I
          SPB       LEAV
          T         DB40.DBW        14                  //save STATUS
          :                                             // :
          :                                             //Error-Handling
          :                                             // :
LEAV:     CLR
```

FC23	Carry out RCV-Receive

```
//      --------------------------------------------
//      Enable Receive Data
//      --------------------------------------------
        SET
        =       DB41.DBX        0.0                 //RCV with EN_R=1
//      --------------------------------------------
//      RCV with Instance-DB
//      --------------------------------------------
        CALL    FB              7 ,       DB70
        EN_R            :=DB41.DBX0.0
        R:=
        LADDR:=DB41.DBW16
        DB_NO:=43
        DBB_NO:=0
        L_TYP:=
        L_NO:=
        L_OFFSET:=
        L_CF_BYT:=
        L_CF_BIT:=
        NDR             :=DB41.DBX0.4
        ERROR            :=DB41.DBX0.5
        LEN              :=DB41.DBW10
        STATUS:=DB41.DBW12
//      --------------------------------------------
//      Check “Receive without error“
//      --------------------------------------------
        U       DB41.DBX        0.4                 //NDR ?
        SPBN    CON1                                //if NO
        L       DB41.DBW        6                   //“Receive without
                                                    //error“
        +1                                          //increment counter
        T       DB41.DBW        6
        L       DB41.DBW        10                  //save
        T       DB41.DBW        4                   //Receive-Length LEN
        SPA     LEAV
//      --------------------------------------------
//      Check “Receive with error“
//      --------------------------------------------
CON1:   U       DB41.DBX        0.5                 //ERROR ?
        SPBN    CON2                                //if NO
        L       DB41.DBW        8                   //“Receive with error“
        +1                                          //increment counter
        T       DB41.DBW        8
        L                       0
        L       DB41.DBW        12                  //if STATUS <>0
        ==I
        SPB     LEAV
        T       DB41.DBW        14                  //save STATUS
        SPA     LEAV
//      --------------------------------------------
//      Check “Error in STATUS“
//      --------------------------------------------
CON2:   L 0
        L       DB41.DBW        12                  //if STATUS <>0
        ==I
        SPB     LEAV
        T       DB41.DBW        14                  //save STATUS
        LEAV:   CLR
```

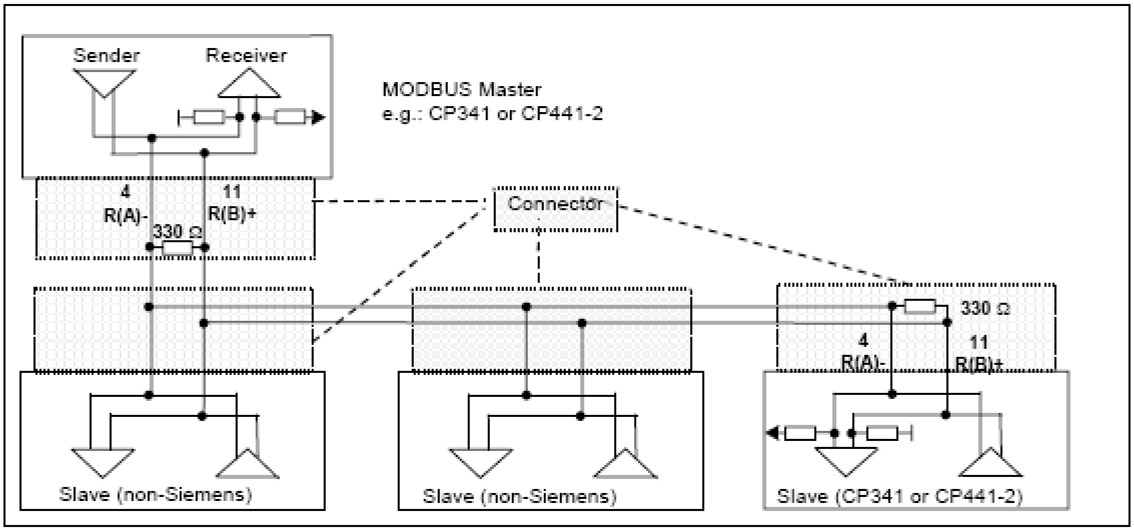
A Wiring Diagrams Multipoint

Wiring diagram RS422 multipoint (Modbus Multipoint)



Caution
In the **RS422** mode CP341 can **only be used as a Master** because the transmitter (Sender) always drives the line and never goes to the high-impedance “Tri State“ mode!

Wiring diagram RS485 multipoint (Modbus Multipoint)



The following applies:

- GND (PIN 8 must always be connected on both sides
- The casing shield must be installed everywhere
- A terminating resistor of approx. 330 Ω is to be soldered into the connector on the last receiver of a node sequence
- Recommended cable type: LIYCY 3 x 2 x 0,14 R(A)/R(B) and T(A)/T(B) twisted pairs. For additional information see the "Cables" section of the "Modbus over Serial Line Specification and Implementation Guide" available at www.modbus.org.
- A wiring with "Stub" is not allowed

Wiring diagram RS232 Point to Point (Modbus RS232)

Please refer to Section B.1 of the CP 341 Point – to – Point Communication Manual.

B Literature List

Modbus Protocol

Modbus over Serial Line
Specification & Implementation Guide
V1.0
12/02/02

Modbus Application Protocol Specification
V1.1a
6/4/04

http://www.modbus.org

Glossary

A

Address The address identifies a physical storage location and enables the user to directly access the operand store there.

B

Block Blocks are elements of the user program which are defined by their function, structure, or purpose. With STEP 7 there are
_ Code blocks (FB, FC, OB, SFB, SFC)
_ Data blocks (DB, SDB)
_ User-defined data types (UDT)

Block Call A block call occurs when program processing branches to the called block

Block Parameter Block parameters are wildcards within multiple-use blocks, which are replaced with current values when the relevant block is called.

C

Communications Processor Communications processors are modules for point-to-point connections and bus connections.

Configuration The configuration is the setup of individual modules of the PLC in the configuration table.

CPU Central processing unit of the S7 programmable controller with control and arithmetic unit, memory, operating system, and interfaces to I/O modules.

CRC Cyclic Redundancy-Check = Checksum which guaranteed accuracy of error recognition.

Cycle Time The cycle time is the time the CPU needs to scan the user program once.

Cyclic Program Processing In cyclic program processing, the user program is executed in a constantly-repeating program loop, called a cycle.

D

Data Block (DB)

These are blocks containing data and parameters with which the user program works. Unlike all other blocks, data blocks do not contain instructions. They are subdivided into global data blocks and instance data blocks. The data held in the data blocks can be accessed absolutely or symbolically. Complex data can be stored in structured form.

Data Type

Data types allow users to define how the value of a variable or constant is to be used in the user program. They are subdivided into elementary and structured data types.

Default Setting

The default setting is a practical basic setting, which is always used if no other value is specified.

Diagnostic Buffer

Every CPU has a diagnostic buffer, in which detailed information on diagnostic events is stored in the order in which they occur.

Diagnostic Event

Diagnostic events are, for example, errors on a module or system errors in the CPU, which may be caused by a program error or by operating? mode transitions.

Diagnostics Functions

The diagnostics functions cover the entire system diagnosis and include detection, analysis and reporting of errors within the PLC.

Download

Downloading means loading load objects (e.g. code blocks) from the programming device into the load memory of the CPU.

F

Function Block (FB)

Function blocks are components of the user program and, in accordance with the IEC standard, are ”blocks with memory”. The memory for the function block is an assigned data block of the ”instance data block”. Function blocks can be assigned parameters, or they can be used without parameters.

H

Hardware

Hardware is the term given to all the physical and technical equipment of a PLC.

I

Instance Data Block

An instance data block is a block assigned to a function block and contains data for this particular function block.

Interface Submodule

The CP 441-2 interface submodule is responsible for the physical conversion of signals. By changing the interface submodule, you can make the communications processor compatible with the communications partner.

Interrupt

An interrupt occurs when program processing in the processor of a PLC is interrupted by an external alarm.

M

Module

Modules are pluggable printed circuit boards for programmable controllers.

Module Parameter

Module parameters are used to set the module reactions. A distinction is made between static and dynamic module parameters.

O

Online/Offline

Online means that a data circuit exists between PLC and programming device. Offline means that no such data circuit exists.

Online Help

STEP 7 allows you to display contextual help texts on the screen while you are working with the programming software.

Operand

An operand is part of a STEP 7 instruction and states with what the processor is to do something. It can be both absolutely and symbolically addressed.

Operating Mode

The SIMATIC S7 programmable controllers have three different operating modes: STOP, RESTART and RUN. The functionality of the CPUs varies in the individual operating modes.

Operating System of the CPU

The operating system of the CPU organizes all functions and operations of the CPU which are not connected to a specific control task.

P

Parameter Parameters are values that can be assigned. A distinction is made between block parameters and module parameters.

Parameter Assignment Parameter assignment means setting the behavior of a module.

Parameter Assignment Tool CP: Point-to-Point Communication, Parameter Assignment The CP Point-to-Point Communication, Parameter Assignment Tool is used to assign parameters to the interface submodule of the communications processor and to set the driver-specific parameters. The standard range is expanded for each loadable driver.

Point-to-Point Connection In a point-to-point connection the communications processor forms the interface between a PLC and a communications partner.

Procedure The execution of a data interchange operation according to a specific protocol is called a procedure.

Process Image The process image is a special memory area in the PLC. At the beginning of the cyclic program, the signal states of the input modules are transferred to the process image input table. At the end of the cyclic program, the process image output table is transferred to the output modules as signal state.

Programmable Controller Programmable controllers (PLCs) are electronic control devices consisting of at least one central processing unit, various input/output modules, and operator control and monitoring devices.

Project Configuration of Data Link Project configuration of data link is the term given to the allocation of a Connection ID in the system function block. The Connection ID enables the system function blocks to communicate between two communication terminal points.

Protocol The communications partners involved in a data interchange must abide by fixed rules for handling and implementing the data traffic. These rules are called protocols.

R

Rack A rack is the rail containing slots for mounting modules.

RESTART On transition from the STOP to the RUN mode, the PLC goes through the RESTART mode.

S

Software

Software is the term given to all programs used on a computer system. These include the operating system and the user programs.

Standard Mode

The standard mode of Modbus ASCII slave driver means, that the parameter “with 32-Bit registers” is not set. In this mode all registers imply 16-bit values.

STEP 7

This is the programming software for SIMATIC S7 programmable controllers.

System Block

System blocks differ from the other blocks in that they are already integrated into the S7-300/400 system and are available for already defined system functions. They are subdivided into system data blocks, system functions, and system function blocks.

System Function (SFC)

System functions are modules without memory which are already integrated into the operating system of the CPU and can be called up by the user as required.

System Function Block (SFB)

System function blocks are modules with memory which are already integrated into the operating system of the CPU and can be called up by the user as required.

U

Upload

Uploading means loading load objects (e.g. code blocks) from the load memory of the CPU into the programming device.

User Program

The user program contains all instructions and declarations for signal processing, by means of which a system or a process can be controlled. The user program for SIMATIC S7 is structured and is divided into smaller units called blocks.

V

Variable

A variable is an operand (e.g. E 1.0), which can have a symbolic name and can therefore also be addressed symbolically.

W

With 32-Bit Registers

When choosing “with 32-Bit Register” mode, holding registers can imply 32-bit values (integer and floating point) as well as 16-bit values when accessed by a master.

Work Memory

The work memory is a RAM on the CPU, which the processor accesses while processing the user program.