

Driver building blocks for
the communication between
Siemens Simatic S7 PLC
and Berger Lahr TLC5xx
Twin Line series controllers
via Profibus-DP

PR 649.00

Version 1.007

Author : Berger Lahr, Dept. TS
Release : Jan 04, 2001

Revision : May 24, 2002

Table of contents

1	IMPORTANT GENERAL INFORMATION	5
1.1	Availability	5
1.1.1	Homepage	5
1.1.2	Hotline	5
2	GENERAL DESCRIPTION	6
2.1	Application	6
2.2	Safety information	6
2.3	Prerequisites	6
2.4	System Description	6
2.4.1	Information on this manual	6
2.4.2	Function	7
2.4.3	Installation	7
2.4.4	Information on the Instruction List code of the driver	7
2.4.5	Building blocks	8
2.4.5.1	General building blocks (minimum version)	8
2.4.5.2	Servomotor building blocks	8
2.4.5.3	Stepping motor building blocks	9
2.4.5.4	Organization blocks	9
2.4.6	Setting of additional parameters	10
2.4.7	Explanations to the application	11
2.4.8	Example	11
2.4.9	Overview of the building blocks	11
3	PROGRAMMING GUIDELINES	13
3.1	Naming conventions	13
3.2	Data type designation	13
3.3	Designation of array variables	13
3.4	Input, output and local variables of building blocks and functions	14
4	DESCRIPTION AND STRUCTURE OF THE BUILDING BLOCKS	15
4.1	Symbol table	15
4.2	Data blocks DB40 (Data_Example1_SERVO) and DB50 (Data_Example1_SM)	15
4.2.1	General description	15
4.2.2	Technical specifications	15
4.3	Organization block OB1 (Main_Task)	15
4.3.1	General description	15
4.3.2	Technical specifications	15

4.4	Organization block OB100 (Init_Task)	15
4.4.1	General description	16
4.4.2	Technical specifications	16
4.5	Function FC30 (Berger_COMMAND)	17
4.5.1	General description	17
4.5.2	Transmit parameters to the function	17
4.5.3	Return values of the function	17
4.5.4	Examples	18
4.5.5	Technical specifications	18
4.6	Function FC31 (DONE)	19
4.6.1	General description	19
4.6.2	Transmit parameters to the function	19
4.6.3	Return values of the function	19
4.6.4	Examples	20
4.6.5	Technical specifications	20
4.7	Function FC32 (Cycl_Update)	20
4.7.1	General description	21
4.7.2	Transmit parameters to the function	21
4.7.3	Return values of the function	21
4.7.4	Example	21
4.7.5	Technical specifications	21
4.8	Function FC 33 (STATE)	21
4.8.1	General description	22
4.8.2	Transmit values to the function	22
4.8.3	Return values of the function	22
4.8.4	Example	22
4.8.5	Technical specifications	23
4.9	Function FC 34 (CHECK_ERROR)	23
4.9.1	General description	23
4.9.2	Transmit values to the function	24
4.9.3	Return values of the function	24
4.9.4	Example	24
4.9.5	Technical specifications	24
4.10	Function FC35 (INIT_FIELDBUS)	25
4.10.1	General description	25
4.10.2	Transmit parameters to the function block	25
4.10.3	Example	25
4.10.4	Technical specifications	25
4.11	Function FC37 (FAULT_RESET)	26
4.11.1	General description	26
4.11.2	Transmit values to the function	26
4.11.3	Return values of the function	26
4.11.4	Example	26
4.11.5	Technical specifications	26
4.12	Function blocks FB40 (Example1_SERVO) and FB50 (Example1_SM)	27
4.12.1	General description	27
4.12.2	Transmit and return values of the function	27

5	EXAMPLE PROJECT	28
----------	------------------------	-----------

5.1	Initialization	28
5.2	Main program	28
5.3	Initialization and positioning of the Berger Lahr Controller _Servomotor	29
5.4	Initialization and positioning of the Berger Lahr Controller _Stepping motor	29
6	ERROR DESCRIPTION	30
6.1.1	General description	30
6.1.2	Error codes in the Parameter.Errorcode (DIW 30)	30
6.1.3	Error detection during initialization or positioning	30
6.1.4	Error word description	30

1 Important General Information

1.1 Availability

Berger Lahr is available in several ways in order to offer you help, support and product information.

1.1.1 Homepage

The Berger Lahr URL is:

<http://www.berger-lahr.de>

For downloads the following address is available for you:

<http://www2.sig-positec.de/public/download.nsf>

1.1.2 Hotline

The BERGER LAHR hotline can be reached via the following phone numbers:

In Germany: 07808 / 943 226
International: xx49 / 7821 / 943 226

Our experts will answer your questions during the following office hours:

	Morning	Afternoon
Monday - Friday	8 AM – 12 PM	1 PM – 5 PM

The hotline can also be reached via fax:

In Germany: 07808 / 943 499
International: xx49 / 7808 / 943 499

Here you will be able to get support concerning the driver building blocks.

2 General Description

2.1 Application

This application provides driver building blocks, which facilitate to a great extent communication between BERGER LAHR Twin Line TLC 5xx series controllers and a Siemens Simatic S7 master PLC via Profibus-DP.

The driver building blocks were created in the Siemens S7 programming language STEP7 © (version 5.0) and combined into a sample project. This project serves as the basis for the development of control software to control BERGER LAHR Twin Line TLC 5xx series controllers via Profibus-DP interface.

2.2 Safety information

This manual is intended to provide extended information in addition to that offered in the manuals of the standard devices. All safety information contained in the standard manuals must be observed.

The driver building blocks for the communication via Profibus-DP between the Siemens Simatic S7 PLC and Berger Lahr controllers of the Twin Line series TLCxxx consist of the software PR 649. Only trained specialists must install this software. Upon request (and without any obligation) we can give you a list of companies who already installed the software without any (as far as we know) complaints. The installation must be strictly carried out in compliance with our installation instructions. Non-observance of the installation instructions can cause damage to humans and the machine.

2.3 Prerequisites

In order to be able to use the driver building blocks successfully, the user must have at least the following components:

- A Siemens Simatic S7 PLC with integrated Profibus-DP master interface (e.g. **S7-315-2 DP**, **S7-413-2 DP**, **S7-414-2 DP**, **S7-416-2 DP**) with the following feature:
⇒ at least **2,12kByte** available memory in the PLC in order to use the minimum version of the driver
- Programming device (such as PG720, PG740) for the Siemens Simatic S7 PLC
- Berger Lahr Twin Line TLC 5xx series controllers, the driver can be used for servo drives (TLC 53x) as well as stepping motor drives (TLC 51x) of different power classes.

2.4 System Description

2.4.1 Information on this manual

This manual is intended to complement the standard manuals for the devices and the manual on online command processing for Twin Line devices. It describes the functions, the installation and the application of the driver building blocks.

Make sure to work through the manuals on the devices and on online command processing for Twin Line devices before reading this manual.

The manuals are listed below with individual ordering numbers, so they can be ordered separately if required.

(The information in this application manual is valid and supersedes other information).

Standard device manuals	Ordering no.
-------------------------	--------------

Standard device manuals	Ordering no.
TLC 51x	98441113118
TLC 53x	98441113110

Standard device manual for Online-Command processing	Ordering no.
Profibus-DP	98441113126

2.4.2 Function

The user controls and operates one or more Berger Lahr Twin Line device series controllers with a Profibus-DP system. The Profibus-DP master and the Berger Lahr controller(s) communicate via a defined data protocol, which is described in the standard manual on online command processing for Profibus-DP.

These driver building blocks facilitate the application work of the user with the data protocol. In addition, an initialization routine is provided. It allows the initialization of the corresponding axis. The only left to do for the user is to set the parameters for this program sequence.

The initialization routine is not required if the Twin Line Control Tool (TLCT) is used to set the parameters.

Multiple examples illustrate the execution of a positioning sequence. **However, for safety reasons there is no reference move described.**

The drive is enabled after an error-free initialization sequence. Therefore the driver building block is independent of the type of the signal interface setting (settings.IO_mode). The drive must be disabled before executing the driver building block because several parameters are transmitted to the device than can only be manipulated in the disabled state.

2.4.3 Installation

The driver building blocks are saved as an S7 archive project on a floppy disk.

When the project is downloaded via URL, the user will receive a self-extracting EXE file. Simply execute the EXE file. The following files should then be available:

- **"649010xx.ARJ" :**
S7 archive containing the sample project. Run the Simatic Manager and choose the menu and function: **File, Extract**

After you have decompressed the sample project, the entire sample project files are available.

- **"649010xe.doc" :**
Contains this manual in the Word for Windows 97 format.

2.4.4 Information on the Instruction List code of the driver

Please note that these driver building blocks were created in instruction list. A correct representation in contact plan (ladder logic) or function plan cannot be guaranteed.

2.4.5 Building blocks

2.4.5.1 General building blocks (minimum version)

The following building blocks must be contained in the project:

Type + No.	Name	Description
FC30	Berger_COMMAND	This central function controls the entire data protocol with the Berger Lahr controller. It sends the desired command to the controller, checks whether the command was executed properly and handles the errors.
FC31	DONE	This function verifies that an action command was executed and correctly terminated.
FC32	CYCL_UPDATE	This block periodically updates the field bus data in the corresponding instance data block.
FC33	STATE	This function evaluates and checks the current state of the TLC state machine and waits until the set condition is reached.
FC34	CHECK_ERROR	This function evaluates and processes error messages and generates the error in the corresponding instance data block.
FC35	INIT_FIELDBUS	This building block initialises the data building block for a Twin Line controller.

Table 1: Minimum version of the driver

These six building blocks constitute the **minimum version** of the drivers, i.e. they are mandatory for operating a Berger Lahr single- or multi-axis controller in a field bus environment. The building blocks require **2,12kByte** of PLC **operating memory**.

These blocks are applicable independently of the drive technology (stepper drive or servo drive).

2.4.5.1.1 Supplementary blocks

Type + No.	Name	Description
FC37	Fault_Reset	This function clears the error memory and generates a Fault_Reset on the positioning controllers. The block <i>can</i> be used optionally. In the example program it is embedded in OB 1.

Table 2: Supplementary block

2.4.5.2 Servomotor building blocks

Type + No.	Name	Description
FB40	EXAMPLE1_SERVO	This function block contains an example program (initialization and positioning) for a single axis servo motor controller (TLC 53x)
DB40	Data_Example1_SERVO	Instance data block that is assigned to the function FB40.
FC40	INIT_SERVO	This function contains the complete initializing sequence for a servo motor axis.
FC41	POS_SEQ_SERVO	Example for a position sequence of a Servo motor axis.

Table 3: Blocks for the field bus operation of a servo motor device.

These four blocks provide a complete and executable initialization and positioning sequence for a servo motor axis. The building blocks require **3,2kByte** of additional operating memory.

2.4.5.3 Stepping motor building blocks

Type + No.	Name	Description
FB50	EXAMPLE1_SM	This function block contains an example program (initialization and positioning) for a single axis stepper motor controller (TLC 51x)
DB50	Data_Example1_SM	Instance data block that is assigned to the function FB50.
FC50	INIT_SM	This function contains the complete initializing sequence for a stepper motor axis.
FC51	POS_SEQ_SM	Example for a position sequence of a stepping motor axis.

Table 4 : Blocks for the field bus operation of a stepping motor device.

These four blocks provide a complete and executable initialization and positioning sequence for a stepping motor axis. The building blocks require **3,2kByte** of additional operating memory.

2.4.5.4 Organization blocks

Type + No.	Name	Description
OB100	INIT_TASK	Example of an initialization of two Berger Lahr controllers after power on and start of the S7.
OB1	MAIN_TASK	Main program
UDT101	DEVICE_DATA	Data type definition for a Berger Lahr controller.

Table 5: Organization blocks

These blocks are mandatory for the driver operation.

2.4.6 Setting of additional parameters

To be able to communicate with a Twin Line controller via Profibus-DP,

1. the Profibus address must be set at the Berger Lahr controller.
2. the Twin Line controller device configuration files must be included in the STEP7 hardware configuration.
3. the Profibus must be configured.

Please refer to the device manuals and the STEP7 manual for details.
The current device configuration file is saved on the documentation CD.

2.4.7 Explanations to the application

The application represents a structure into which the user can embed his application. FB 40 and FB 50 are used as the interface in conjunction with corresponding axis related instance data blocks. For this, it is possible to expand the case instruction in network 2 by adding different positioning sequences (see FC41/FC51). Incrementing the jump variables can be application dependent (e.g. in additionally inserted jump labels). Alternatively the implemented positioning sequence (FC41/FC51) can be modified accordingly.

2.4.8 Example

The example describes, how functions can be embedded in the structure. (The inserted program parts are displayed *italic-bold*)

```
///Example to embed other functions in the structure
Network 2:
    L DIW 24
    SPL LIST                                // Jump list
    SPA INI0
    SPA POS0                                // 1. Positioning sequence
    SPA GO_1                                // Switching condition
    SPA POS2                                // 2. Positioning sequence
    SPA GO_2                                //....
    SPA POS3
    SPA GO_3
    SPA ...
    SPA ...
LIST:    SPA END                            // Jump to end

...
Network 3:

Network 5:

GO_1:    NOP      0
          U        E 0.0                    // And instruction with input E 0.0 and flag M 20.0
          U        M 20.0
          SPB      NSTP                    // If the condition is true go to the next step

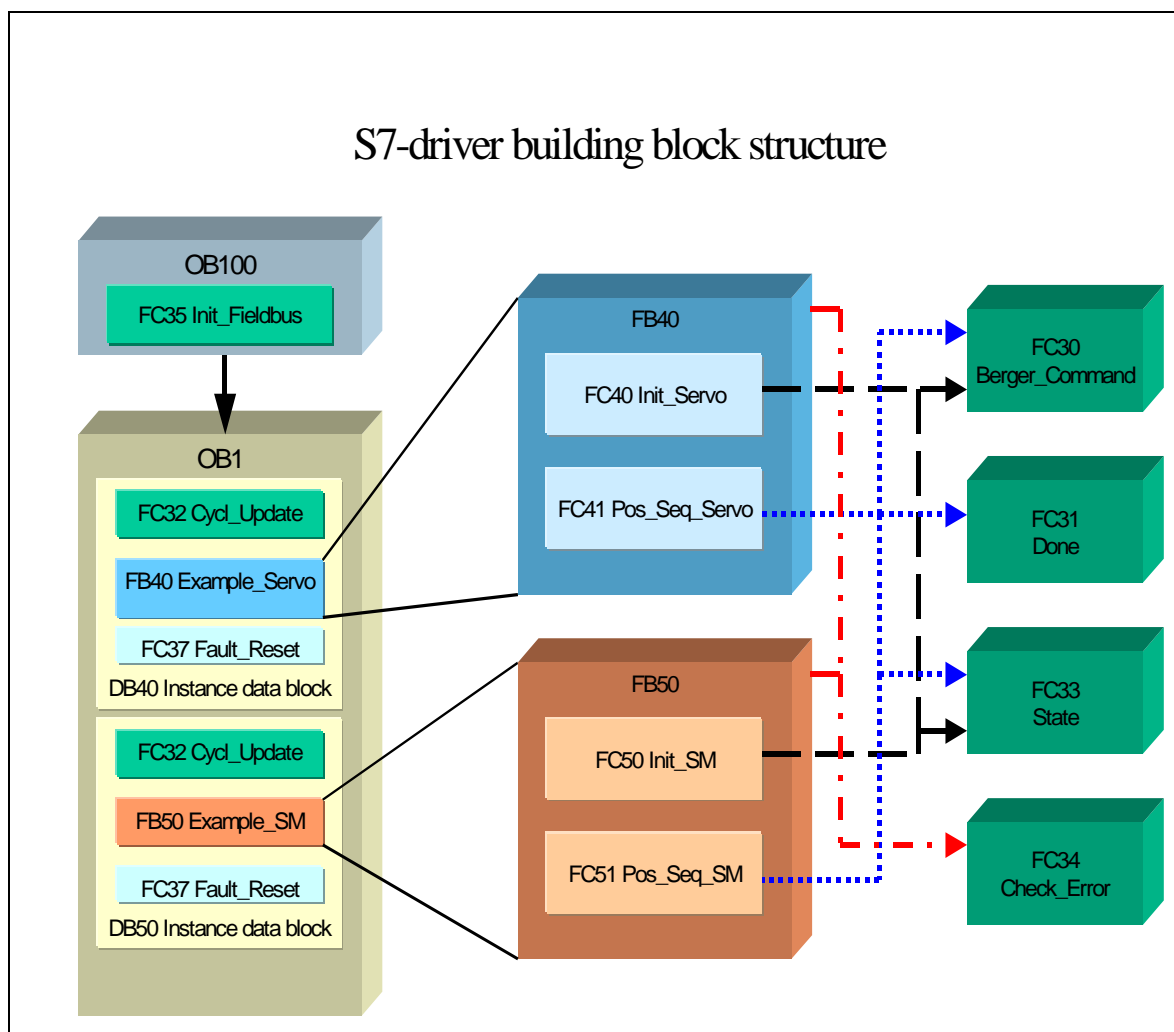
          SPA Err                            // Error in the function

///;
```

2.4.9 Overview of the building blocks

The chart below shows all building blocks in the S7-project. Arrows indicate how the individual building blocks are called and how they are linked.

Graphic 1: System overview



3 Programming Guidelines

The naming conventions described below are used in this manual to further facilitate your work with variables in the STEP7 programming language.

3.1 Naming conventions

The names of variables are always preceded by a code consisting of two lowercase letters. This code identifies the data type. The actual name of a variable always begins with an uppercase letter.

Advantages:

- easy identification of the data type
- improved legibility of the code

Therefore, variables always have this name structure:

- typeName

3.2 Data type designation

Data type	Code	Example
BOOL	bo	boStart
BYTE	by	byValue
WORD	wo	woWord
DWORD	dw	dwDoubleword
INT	in	inVariable
DINT	di	diPosition
REAL	re	reReal
DATE	da	daDate
TIME	ti	tiTimer
S5TIME	t5	t5Timer
TIME_OF_DAY	td	tdTimer
CHAR	ch	chCharacter
STRING	st	stBuffer
STRUCT	sc	scStructure

3.3 Designation of array variables

An array is a field of variables of a simple data type. The data type of an array variable is preceded by "**ar**", so that the name structure of an array variable is:

artypName

example: **arinStatus** = array variable "Status" of type integer

3.4 Input, output and local variables of building blocks and functions

Data type and type of the variable should be indicated by the name. The variables are preceded by the following codes:

Data type	Code	Example
IN	itype	iinValue = input variable "Value" of type INTEGER
OUT	otype	oboErr = output variable "Err" of type BOOLEAN
IN_OUT	ioype	ioreValue = IN_OUT variable "Value" of type REAL
TEMP and STAT	type	woStatus = local building block variable "Status" of type WORD

4 Description and structure of the building blocks

4.1 Symbol table

In the symbol table, the flag memory areas used by the driver building blocks receive symbolic names. These flag memory areas must not be used by the user application.

In addition, symbolic names are entered for the functions building blocks, the functions and the data building blocks.

4.2 Data blocks DB40 (Data_Example1_SERVO) and DB50 (Data_Example1_SM)

4.2.1 General description

These data blocks are instance data blocks that are assigned to the accompanying function blocks with the same number (e.g. FB40, DB40). They contain all necessary data to be able to operate one Berger Lahr field bus controller per data block. The number of the data blocks must not be changed. An additional function block and an accompanying instance data block must be created if additional Berger Lahr controllers should be operated on the field bus. The required structure is saved in the UDT101.

4.2.2 Technical specifications

Data:	80 Bytes
Load memory requirement:	272 Bytes
Main memory requirement:	116 Bytes
Version:	01.06

4.3 Organization block OB1 (Main_Task)

4.3.1 General description

This block organizes the main program. The blocks FB 40 and FB 50 as well as FC 37 and FC 32 are called in the example project. A more detailed description can be found in *chapter 5.2*.

4.3.2 Technical specifications

Main memory:	198 Bytes
MC7-Code:	162 Bytes
Local data:	30 Bytes
Version:	01.06

4.4 Organization block OB100 (Init_Task)

4.4.1 General description

This block becomes active at the CPU start-up. The example project contains FC 35 for device initialization (for a more detailed description see *chapter 5.1* initialization)

4.4.2 Technical specifications

Main memory: 122 Bytes

MC7-Code: 86.Bytes

Local data: 26 Bytes

Version: 01.06

4.5 Function FC30 (Berger_COMMAND)

4.5.1 General description

This function is the central building block called by the master building blocks (FC40, FC41, FC50,...). It writes the required data to the field bus, reads the controller feedback from the field bus, checks the feedback and generates a global error code. It must be considered that the write command must be triggered only once.

The controller periodically returns the desired read value until a new read command is requested.

4.5.2 Transmit parameters to the function

This function handles all the required data in the current instance data building block (e.g. DB40), assigned to the accompanying Berger Lahr controller.

If the user wants to execute a command with this function, he has to specify the following parameters:

- | | |
|---------------|--|
| 1. access : | Enter access type (write = 4 / read = 0) |
| 2. subindex : | Enter sub-index of the desired object |
| 3. index : | Enter index of the desired object |
| 4. data : | Here the data is transmitted to trigger the desired action
or to transmit a defined value. The data is only valid for write commands |

4.5.3 Return values of the function

The function provides the following return values:

- | | |
|-------------------|---|
| 1. oboCmd_Busy: | This flag is set while the command is being executed
Only after this flag was reset and no error occurred, this command was successfully executed or started. |
| 2. oboCmd_Error : | This flag is set if an error occurs in the execution of the command.
In addition, an error code is written to the variable #parameter.Errorcode(DIW 30) of the instance data building block (e.g. DB40). |
| 3. Read data : | Read data is provided in the #Parameter.In3_Word(DIW6) and #Parameter.In4_Word(DIW8) or in the Parameter.Receivedata(DID62) of the accompanying instanceDB. |

4.5.4 Examples

The two examples below illustrate the usage of this function block:

```
;;
CM30:  NOP      0
      CALL "Berger_COMMAND"          //
      Access    :=B#16#4             // Write
      subindex  :=B#16#1B            // Object 29.27
      index     :=W#16#1D            // Deceleration ramp
      data      :=DW#16#1388         // Value: 5000 Rev/(min*sec)
      oboCmd_Busy :=#boCmd_Busy
      oboCmd_Error:=#boCmd_Error
;;
      U        #boCmd_Busy
      SPB      BUSY                  // Function still busy
      UN       #boCmd_Error
      SPB      NSTP                  // Go to the next step
;;
      SPA Err                        // Error in the function
;;
```

Example 1 : Call deceleration ramp command with FC30

In this example the deceleration ramp command is set and started via the function FC30.

```
;;
CM40:  NOP      0
      CALL "Berger_COMMAND"          //
      Access    :=B#16#4             // Write
      subindex  :=B#16#5            // Object 35.5
      index     :=W#16#23            // ptp.v_target (Velocity)
      data      :=DW#16#1F4         // Value: 500 rev/min
      oboCmd_Busy :=#boCmd_Busy
      oboCmd_Error:=#boCmd_Error
;;
      U        #boCmd_Busy
      SPB      BUSY                  // Function is still busy
      UN       #boCmd_Error
      SPB      NSTP                  // Go to the next Step
;;
      SPA Err                        // Error in the Function
;;
```

Example 2 : Call PTP.V_TARGET with FC30

In this example the command PTP.V_TARGET is started via the function FC30.

4.5.5 Technical specifications

Main memory: 570 Bytes

MC7-Code: 534 Bytes

Local data: 22 Bytes

Version: 01.06

4.6 Function FC31 (DONE)

4.6.1 General description

This function checks if an action command was executed and completed successfully:

Depending on the specified parameters, these action commands require a certain processing time (e.g. a relative positioning with the object 35.3 from position 0 to position 5000). During this processing time, other commands can be transmitted to Berger Lahr controllers. FC31 (DONE) can be used in order to check for complete execution of the command.

4.6.2 Transmit parameters to the function

This function building block receives the required parameters of the most recently executed command from the instance data block (e.g. DB40).

4.6.3 Return values of the function

The function provides the following return values:

1. oboDone_Busy : This flag is set as long as the most recently sent command is being executed (e.g. positioning). Only if no error occurred and the flag is reset, the command (e.g. positioning) was successful.
2. oboDone_Error : This flag is set if an error occurs in the execution of the command most recently sent. In addition, an **error code** is written to the variable **#parameter.Errorcode** of the instance data building block (e.g. DB40).

4.6.4 Examples

Below is an application example of this function:

```
Network 6
CM40:      NOP 0                                // Positioning with object 35.1
           CALL „Berger_COMMAND“
           Access      :=B#16#4
           Subindex    :=B#16#1
           Index       :=W#16#23
           Data        :=DW#16#17700
           OboCmd_Busy  :=#boCmd_Busy
           OboCmd_Error:=#boCmd_Error

           U          #boCmd_Busy
           SPB        BUSY                        // Function is still busy
           UN         #boCmd_Error
           SPA        ERR                        // Error in the Function

           SPA NSTP                                // Function executed, go to the next
                                           // Step

Network 7:
CM50:      NOP 0
           CALL "DONE"                          // Monitor the end of the positioning
           oboDone_Busy :=#boDone_Busy
           oboDone_Error:=#boDone_Error

//;
           U          #boDone_Busy
           SPB        BUSY                        // Function is still busy
           UN         #boDone_Error
           SPA        ERR                        // Error in the Function

//;
           SPA NSTP                                // Function executed, go to the next
                                           // Step

//;
```

Example 3 : Calling the DONE function

In the first step a MOVE-command is started. This command starts the axis 1 motor connected to the Berger Lahr controller. In the next step the DONE-function checks if the motor has reached the target position. Only if this is the case and no axis error occurred it will be branched off to the next operating step.

4.6.5 Technical specifications

Main memory: 162 Bytes

MC7-Code: 126 Bytes

Local data: 24 Bytes

Version: 01.06

4.7 Function FC32 (Cycl_Update)

4.7.1 General description

This function periodically reads and updates the data from the Profibus for the accompanying instance data block.

4.7.2 Transmit parameters to the function

No parameters must be passed on to the FC32.

4.7.3 Return values of the function

No return values are provided.

If no data can be read from the field bus, an **error code** is written to the variable **#parameter.Errorcode** of the instance data block (e.g. DB40).

4.7.4 Example

The example below illustrates the call of FC32 in the OB1:

```
Network 1: Example servomotor

//;
On DI 40          // accompanying instance data block is opened
CALL „CYCL_UPDATE“ // FC 32 reading periodically from the Profibus

//;
```

Example 4 : Calling of the FC32

At first the data block with the desired parameter set is opened.
Then the function is called with a CALL. By using the CALL instruction the function is independent of the current result (CR) or a condition.

4.7.5 Technical specifications

This function must be called periodically.

Main memory: 146 Bytes

MC7-Code: 110 Bytes

Local data: 18 Bytes

Version: 01.06

4.8 Function FC 33 (STATE)

4.8.1 General description

This function checks the status of the device state machine.

The value of the state that should be checked is entered, e.g. state 4.

The „**State**“ is now active as long as the desired value is reached.

There is a global error message output if the transmission or the device is in the error state.

If the switching of the state machine takes too long a time-out is generated and an error number is entered in the **#parameter.errorcode**.

The block is especially useful during a device initialization in order to check the device state machine.

4.8.2 Transmit values to the function

The function provides the following transmit value

1. lbyStateNo Here the expected status of the state machine is entered. The range is between 3 → 6

4.8.3 Return values of the function

The function provides the following return values:

1. oboDone_Busy The bit is set to 1 until the expected status is reached.
2. oboDone_Error This bit is set if a transmission error occurs.
3. oboDevice_Error If the device is outside of the conditions 3→6, certain start conditions are missing or the device is in the error state (Status 7,9).in this case it is indicated here (Bit = True)
4. owoStateNo The current status is indicated here. It can be displayed in data-word or in a flag-word.

4.8.4 Example

```
Network 15:
  Enab:    NOP 0
          CALL "Berger_COMMAND"           //
          Access    :=B#16#4              // write
```

```
subindex      :=B#16#1           // Object 28.1
index         :=W#16#1C          // driveCtrl
data          :=DW#16#2          // Value: Enable
oboCmd_Busy   :=#boCmd_Busy
oboCmd_Error  :=#boCmd_Error

//;

U      #boCmd_Busy
SPB    BUSY           // Function is still busy
UN     #boCmd_Error
SPB    NSTP           // go to the next Step
SPA    Err            // Error in the Function

// FC 33 is called, to monitor state 6
Network 16:
Sta6:   NOP 0
        CALL „STATE“
        IbyStateNo      :=B#16#6           // Expected state 6
        OboDone_Busy     :=#boCmd_Busy      // Block active
        OboDone_Error    :=#boDone_Error    // Transmission error
        OboDevice_Error  :=#boCmd_Error     // Device error
        OwoStateNo       :=DIW56

//;

U      #boCmd_Busy           // Block is active

SPB    Busy
UN     #boCmd_Error         //
UN     #boDone_Error
SPB    NSTP                 // go to the next Step
SPA    Error
```

Example 5 : Status 6 is checked after the device was switched on

In the example FC 33 is called to monitor the state 6, after the device was switched on.

4.8.5 Technical specifications

Main memory: 438 Bytes

MC7-Code: 402 Bytes

Local data: 22 Bytes

Version: 01.06

4.9 Function FC 34 (CHECK_ERROR)

4.9.1 General description

This function reads the current errors.

The block structure is according to the error handling of synchronous and asynchronous errors, as described in the Profibus documentation on the pages 7-1 to 7-3.

4.9.2 Transmit values to the function

No parameters must be passed on to the function.

4.9.3 Return values of the function

1. oboErrCheck_Busy As long as the variable is *True* the block is processed, if *False* the processing is finished
2. oinErrCheck_Error If an error occurs while processing the function, the oboErrCheck_Busy is set to *False* and an error number is entered here, indicating where the error occurred.

4.9.4 Example

```
;;
POS0:  NOP 0
      CALL "POS_SEQ_SERVO"
      OboSeq_Busy :=#boSeq_Busy // Block is active
      OinSeq_Error :=#inSeq_Error // Error no. where positioning is interrupted

;;
      U #boSeq_Busy
      SPB BUSY // Function is still busy

      L #inSeq_Error // Check if error
      L 0
      <>I
      SPB Err
      SPA NSTP // go to the next step

;;
Err:   NOP 0
      CALL "CHECK_ERROR"
      OboErrCheck_Busy :=#boErrCheck_Busy
      OinErrCheck_Error :=#inErrCheck_Error

      U #boErrorCheck_Busy
      SPB Busy
      L #inErrorCheck_Error
      L 0
      ==I
      SPB SEQE
      L #inErrorCheck_Error
      T DIW38
      SPA END

;;
SEQE:  NOP 0
      R "M_InitServo"
      S "M_ErrorServo"
      SPA END
```

Example 6 : Error evaluation with FC34

This example monitors a positioning sequence for errors and the errors are evaluated via FC 34 (CHECK_ERROR).

4.9.5 Technical specifications

Main memory: 712 Bytes

MC7-Code: 676 Bytes

Local data: 16 Bytes

Version: 01.06

4.10 Function FC35 (INIT_FIELDBUS)

4.10.1 General description

This function initializes the currently opened data building block for a Berger Lahr Controller with the default values and stores the field bus start address for input and output data.

4.10.2 Transmit parameters to the function block

The following two parameters must be passed on to the FC35:

1. iinIADR: Profibus-DP start address for the Berger Lahr controller input data;
value range **0... depending on used CPU**
2. iinOADR: Profibus-DP start address for the Berger Lahr controller output data;
value range **0... depending on used CPU**

4.10.3 Example

The example below shows how FC35 is called in the organisation building blocks OB100 or OB101 (S7 PLC initialization building blocks):

```
Network 1 :  
//;  
ON      DI 40                // DB40 is opened  
CALL    "INIT_FIELDBUS"      // initialize data block  
    iinIADR:=256              // field bus start address for inputs  
    iinOADR:=256              // field bus-start address for outputs  
//;
```

Example 7 : Calling the FC35

First, the instance data building block DB40 is opened. Data block DB40 is now preset with default values when FC35 is called. The field bus start address 256 for the controller input and output data is also stored in this data block.

4.10.4 Technical specifications

Main memory: 174 Bytes

MC7-Code: 138 Bytes

Local data: 0 Bytes

Version: 01.06

4.11 Function FC37 (FAULT_RESET)

4.11.1 General description

This block is used to erase all error memories and to send a Fault_Reset to the accompanying device. The function can be started via a flag or input. The FC 37 can be implemented but is not mandatory.

4.11.2 Transmit values to the function

1. iboCmd_Fault_Reset The function is started if this input variable is set to *True*. Flags or inputs can be used.

4.11.3 Return values of the function

1. oinSeq_Error If an error occurs while processing the block, an error number is entered here, indicating where the error occurred
2. oboInit_Busy As long as the block is active the Boolean variable is set *True*.

4.11.4 Example

```
;;  
  
On DI 40                                     // accompanying instance DB is opened  
CALL „FAULT_RESET“                        //  
  IboCmd_Fault_Reset := "Fault_Reset_Servo" // Flag that start the function  
  OinSeq_Error       := #inSeq_Error  
  OboInit_Busy       := #boInit_Busy
```

Example 8 : FC 37 embedded in the OB1

In this example the FC 37 is embedded in the OB1 and before that the accompanying instance data block is opened.

4.11.5 Technical specifications

Main memory: 394 Bytes

MC7-Code: 358 Bytes

Local data: 12 Bytes

Version: 01.06

4.12 Function blocks FB40 (Example1_SERVO) and FB50 (Example1_SM)

4.12.1 General description

These function blocks represent the frame for an application. E.g. an initialization and a positioning sequence is called here. Depending on the conditions, the user can integrate one or more parameter and positioning sequences into the block. Each axis is assigned to a function block and therefore also to an instance data block. With this an unambiguous assignment between the sequence in the function block and the accompanying data is guaranteed. The disadvantage of this data structure is in the absolute addressing of all data that is referenced via the instance data block. If an additional variable (input variable, output variable and input/output variable) is added, the absolute addresses are shifted. With this all the addresses used in the drivers must be changed. To avoid this, a data exchange can take place, either via the flag area or better via additional data words in the corresponding instance data block.

4.12.2 Transmit and return values of the function

This function exchanges all necessary data via the accompanying instance data block.

5 Example project

This chapter explains how the structure and the programming of a control program can look like. The here described example project is used for this. In this project two Berger Lahr controllers (TLC53x and TLC51x) are operated completely independent from each other. The PLC used is the S7-315-2 with Profibus-DP master interface.

5.1 Initialization

In the organization blocks OB100 both Berger Lahr controllers are initialized.

```
;;  
ON      "Data_Example1_Servo"      // open DB40  
CALL    "INIT_FIELBUS"             // initialize DB40  
    iinIADR:=256                    // Input field bus start address  
    iinOADR:=256                    // Output field bus start address  
;;  
ON      "Data_Example1_SM"         // open DB50  
CALL    "INIT_FIELBUS"             // initialize DB50  
    iinIADR:=264                    // Input field bus start address  
    iinOADR:=264                    // Output field bus start address
```

Source code listing 1 : OB100

In this example project both instance data blocks for the Berger Lahr controllers are initialized. The field bus start address for the input and output data range is 256 for the first controller and 264 for the second controller.

5.2 Main program

In the example project two function blocks (FB40 and FB50) are called in the OB1. FB40 with instance data block DB40 controls the first Berger Lahr controller and FB50 with instance data block DB50 controls the second Berger Lahr controller.

```
;;  
CALL    "Example1_Servo" , "Data_Example1_Servo" // FB40 , execute DB40  
;;  
CALL    "Example1_SM" , "Data_Example1_SM" // FB50 , execute DB50  
;;
```

Source code listing 2 : OB1

If additional Berger Lahr controllers should be operated on the Profibus, it is necessary to program an additional FB with accompanying instance data block per controller. These blocks must also be integrated in the example project in OB1 and OB100 accordingly.

5.3 Initialization and positioning of the Berger Lahr Controller_Servomotor

Programmed in **FB40** of the example project are an initialization routine and a positioning sequence for the Berger Lahr Controller 1_Servo (TLC53x).

Setting the flag M100.0 starts the processing of the FB. First the motor axis is initialized (FC40 initialization) and then it moves 96000 steps absolute CW, then 96000 steps relative and after that back to the position 0.

Then Flag M100.0 is reset.(FC41 positioning)

If an error occurs flag M100.0 is reset flag M100.7 is set as an error message. an error number can be read in **#parameter.Fault_Step** of the accompanying instance data block. M100.6 is set to delete the error. This deletes the error memory and performs a Fault_Reset of the positioning controller. After this the positioning can be started again with M100.0.

M100.0 ⇒ Execute initialization and positioning sequence

M100.6 ⇒ Error reset

M100.7 ⇒ Error indication



NOTE

Due to its considerable length, the source code listing is not shown in this manual. If required, you can print the function building block via STEP7.

5.4 Initialization and positioning of the Berger Lahr Controller _Stepping motor

Programmed in **FB50** of the example project are an initialization routine and a positioning sequence for the Berger Lahr Controller_SM (TLC51x).

Setting the flag M101.0 starts the processing of the FB. The sequence of the initialization routine and positioning sequence (FC 50 and FC 51) is identical to the one for the servomotor. The differentiation is in the different flags.

M101.0 ⇒ Execute initialization and positioning sequence

M101.6 ⇒ Error reset

M101.7 ⇒ Error indication



NOTE

Due to its considerable length, the source code listing is not shown in this manual. If required, you can print the function building block via STEP7.

6 Error description

6.1.1 General description

The error in the example is realized via FC 34.

The function is called in the jump label **ERR** of the function blocks FB 40 and FB 50.

A sequence number is stored into an output variable in case an error occurs during the initialization or positioning. This number is evaluated in the FB. If the variable is not equal to zero an error bit is set and FC 34 "Check Error" is executed. This block itself reads out the error and assigns it to the corresponding data words of the respective axis.

6.1.2 Error codes in the Parameter.Errorcode (DIW 30)

At each call the functions FC30 (Berger_COMMAND), FC31 (DONE), FC32 (CYCLE_UPDATE) and FC 34 (CHECK_ERROR) generate an error code in case of a error. The error code is stored in the variable **#parameter.Errorcode** of the actual instance data block (e.g. DB40) for the Berger Lahr controller. Always only the last error that occurred is stored.

The table below lists all the possible errors:

Error code in variable #parameter.Errorcode (decimal)	Error code in variable #parameter.Errorcode (hex)	Description / Remark
200	00C8	Timeout in FC 30 (Berger_COMMAND)
204	00CC	Profibus write error
205	00CD	Profibus read error
206	00CE	Timeout in the FC 33 (State)
> 4096	1000	See error table in the manual "Technical Documentation TLC5xx"

6.1.3 Error detection during initialization or positioning

If a synchronous error occurs an output variable is transferred. The following code was implemented to be able to see if the error occurred at initialization, positioning, in the error routine (FC34) or in the Fault_Reset(FC37).

- **Initialization** Code 1xx
- **Positioning** Code 2xx
- **Error routine** Code 3xx
- **Fault_Reset** Code 4xx

Here the first number e.g. 1xx indicated that the error occurred during the initialization (e.g.FC40). The following 2 digits (hexadecimal) indicate at which step e.g. **102** the initialization was interrupted.

The code is stored in the #parameter.Fault_Step of the accompanying instance data blocks.

No additional error detection is required because the respective instanceDB knows the concerned axis.

6.1.4 Error word description

The error words are defined in UDT101. Below is an error word description.

- #parameter.Errorcode (DIW30) Synchronous errors are displayed here
- #parameter.FltSig (DID32) Internal error are displayed here (Object 28:18)
- #parameter.SignSR (DIW36) External errors are displayed here (Object 28:15)
- #parameter.Fault_Step (DIW38) Here the sequence number where the error occurred is entered
- #parameter.INT_Sig_SR (DID40) Here the errors are entered that are read via the object 29.34 (INTSigSR)
- #parameter.Stop_Fault (DIW44) Here the last occurred error is entered
Object 32.7 (Stop_Fault)