YASKAWA

VIPA SPEED7 Library

OPL_SP7-LIB | SW90KS0MA V10.002 | Manual

HB00 | OPL_SP7-LIB | SW90KS0MA V10.002 | en | 18-28

Block library - System Blocks

www.vipa.com/en/service-support/manuals

VIPA CONTROLS

VIPA GmbH
Ohmstr. 4
91074 Herzogenaurach
Telephone: +49 9132 744-0
Fax: +49 9132 744-1864
Email: info@vipa.com
Internet: www.vipa.com

SW90KS0MA_000_OPL_SP7-LIB_System Blocks,3,EN -   © 2018

Table of contents

1 General

1.1 Copyright © VIPA GmbH

Every effort has been made to ensure that the information contained in this document was complete and accurate at the time of publishing. Nevertheless, the authors retain the right to modify the information.

This customer document describes all the hardware units and functions known at the present time. Descriptions may be included for units which are not present at the customer site. The exact scope of delivery is described in the respective purchase contract.

CE Conformity Declaration

Hereby, VIPA GmbH declares that the products and systems are in compliance with the essential requirements and other relevant provisions. Conformity is indicated by the CE marking affixed to the product.

Conformity Information

For more information regarding CE marking and Declaration of Conformity (DoC), please contact your local VIPA customer service organization.

Information product support

Contact your local VIPA Customer Service Organization representative if you wish to report errors or questions regarding the contents of this document. If you are unable to locate a customer service centre, contact VIPA as follows:

VIPA GmbH, Ohmstraße 4, 91074 Herzogenaurach, Germany

Telefax: +49 9132 744-1204

EMail: documentation@vipa.de

Technical support

Contact your local VIPA Customer Service Organization representative if you encounter problems with the product or have questions regarding the product. If you are unable to locate a customer service centre, contact VIPA as follows:

VIPA GmbH, Ohmstraße 4, 91074 Herzogenaurach, Germany

Tel.: +49 9132 744-1150 (Hotline)

EMail: support@vipa.de

1.2 About this manual

Objective and contents

The manual describes the block library *'System Blocks'* from VIPA:

- It contains a description of the structure, project implementation and usage in several programming systems.
- The manual is targeted at users who have a background in automation technology.
- The manual is available in electronic form as PDF file. This requires Adobe Acrobat Reader.
- The manual consists of chapters. Every chapter provides a self-contained description of a specific topic.
- The following guides are available in the manual:
  - An overall table of contents at the beginning of the manual
  - References with pages numbers

Icons Headings

Important passages in the text are highlighted by following icons and headings:

DANGER!
Immediate or likely danger. Personal injury is possible.

CAUTION!
Damages to property is likely if these warnings are not heeded.

Supplementary information and useful tips.

2 Important notes

2.1 General

In the following, you will find important notes, which must always be observed when using the blocks.

2.2 Internally used blocks

CAUTION!
The following blocks are used internally and must not be overwritten! The direct call of an internal block leads to errors in the corresponding instance DB! Please always use the corresponding function for the call.

FC/SFC	Designation	Description
FC/SFC 192	CP_S_R	is used internally for FB 7 and FB 8
FC/SFC 196	AG_CNTRL	is used internally for FC 10
FC/SFC 200	AG_GET	is used internally for FB/SFB 14
FC/SFC 201	AG_PUT	is used internally for FB/SFB 15
FC/SFC 202	AG_BSEND	is used internally for FB/SFB 12
FC/SFC 203	AG_BRCV	is used internally for FB/SFB 13
FC/SFC 204	IP_CONF	is used internally for FB 55 IP_CONF
FC/SFC 205	AG_SEND	is used internally for FC 5 AG_SEND
FC/SFC 206	AG_RECV	is used internally for FC 6 AG_RECV
FC/SFC 253	IBS_ACCESS	is used internally for SPEED bus INTERBUS masters
SFB 238	EC_RWOD	is used internally for EtherCAT Communication
SFB 239	FUNC	is used internally for FB 240, FB 241

3 Include library

Block library *'System Blocks'*

The block library can be found for download in the *'Service/Support'* area of www.vipa.com at *'Downloads ➔ VIPA Lib'* as *'Block library System Blocks - SW90KS0MA'*. The library is available as packed zip file. As soon as you want to use these blocks you have to import them into your project.

The following block libraries are available

File	Description
SystemBlocks_S7_V0003.zip	■ Block library for Siemens SIMATIC Manager. ■ For use in CPUs from VIPA or S7-300 CPUs from Siemens.

3.1 Integration into Siemens SIMATIC Manager

Overview

The integration into the Siemens SIMATIC Manager requires the following steps:

1. Load ZIP file
2. "Retrieve" the library
3. Open library and transfer blocks into the project

Load ZIP file

➔ Navigate on the web page to the desired ZIP file, load and store it in your work directory.

Retrieve library

1. Start the Siemens SIMATIC Manager with your project.
2. Open the dialog window for ZIP file selection via *'File ➔ Retrieve'*.
3. Select the according ZIP file and click at [Open].
4. Select a destination folder where the blocks are to be stored.
5. Start the extraction with [OK].

Open library and transfer blocks into the project

1. Open the library after the extraction.
2. Open your project and copy the necessary blocks from the library into the directory "blocks" of your project.

   ⇨ Now you have access to the VIPA specific blocks via your user application.

Are FCs used instead of SFCs, so they are supported by the VIPA CPUs starting from firmware 3.6.0.

4 Block parameters

4.1 General and Specific Error Information RET_VAL

Overview

The return value *RET_VAL* of a system function provides one of the following types of error codes:

- A *general error code*, that relates to errors that can occur in anyone SFC.
- A *specific error code*, that relates only to the particular SFC.

Although the data type of the output parameter *RET_VAL* is integer (INT), the error codes for system functions are grouped according to hexadecimal values.

If you want to examine a return value and compare the value with the error codes, then display the error code in hexadecimal format.

RET_VAL (Return value)

The table below shows the structure of a system function error code:

Bit	Description
7 ... 0	Event number or error class and single error
14 ... 8	Bit 14 ... 8 = "0": **Specific error code** The specific error codes are listed in the descriptions of the individual SFCs. Bit 14 ... 8 > "0": **General error code** The possible general error codes are shown
15	Bit 15 = "1": indicates that an error has occurred.

Specific error code

This error code indicates that an error pertaining to a particular system function occurred during execution of the function.

A specific error code consists of the following two numbers:

- Error class between 0 and 7
- Error number between 0 and 15

Bit	Description
3 ... 0	Error number
6 ... 4	Error class
7	Bit 7 = "1"
14 ... 8	Bit 14 ... 8 = "0"
15	Bit 15 = "1": indicates that an error has occurred.

General error codes RET_VAL

The parameter *RET_VAL* of some SFCs only returns general error information. No specific error information is available.

The general error code contains error information that can result from any system function. The general error code consists of the following two numbers:

- A parameter number between 1 and 111, where 1 indicates the first parameter of the SFC that was called, 2 the second etc.
- An event number between 0 and 127. The event number indicates that a synchronous fault has occurred.

Bit	Description
7 ... 0	Event number
14 ... 8	Parameter number
15	Bit 15 = "1": indicates that an error has occurred.

General error codes

The following table explains the general error codes associated with a return value. Error codes are shown as hexadecimal numbers. The x in the code number is only used as a placeholder. The number represents the parameter of the system function that has caused the error.

Error code	Description
8x7Fh	Internal Error. This error code indicates an internal error at parameter x. This error did not result from the actions if the user and he/she can therefore not resolve the error.
8x01h	Illegal syntax detection for an ANY parameter.
8x22h	Area size error when a parameter is being read.
8x23h	Area size error when a parameter is being written. This error code indicates that parameter x is located either partially or fully outside of the operand area or that the length of the bit-field for an ANY-parameter is not divisible by 8.
8x24h	Area size error when a parameter is being read.
8x25h	Area size error when a parameter is being written. This error code indicates that parameter x is located in an area that is illegal for the system function. The description of the respective function specifies the areas that are not permitted for the function.
8x26h	The parameter contains a number that is too high for a time cell. This error code indicates that the time cell specified in parameter x does not exist.
8x27h	The parameter contains a number that is too high for a counter cell (numeric fields of the counter). This error code indicates that the counter cell specified in parameter x does not exist.
8x28h	Orientation error when reading a parameter.
8x29h	Orientation error when writing a parameter. This error code indicates that the reference to parameter x consists of an operand with a bit address that is not equal to 0.
8x30h	The parameter is located in the write-protected global-DB.
8x31h	The parameter is located in the write-protected instance-DB. This error code indicates that parameter x is located in a write-protected data block. If the data block was opened by the system function itself, then the system function will always return a value 8x30h.
8x32h	The parameter contains a DB-number that is too high (number error of the DB).
8x34h	The parameter contains a FC-number that is too high (number error of the FC).

Error code	Description
8x35h	The parameter contains a FB-number that is too high (number error of the FB). This error code indicates that parameter x contains a block number that exceeds the maximum number permitted for block numbers.
8x3Ah	The parameter contains the number of a DB that was not loaded.
8x3Ch	The parameter contains the number of a FC that was not loaded.
8x3Eh	The parameter contains the number of a FB that was not loaded.
8x42h	An access error occurred while the system was busy reading a parameter from the peripheral area of the inputs.
8x43h	An access error occurred while the system was busy writing a parameter into den peripheral area of the outputs.
8x44h	Error during the n-th (n > 1) read access after an error has occurred.
8x45h	Error during the n-th (n > 1) write access after an error has occurred. This error code indicates that access was denied to the requested parameter.

5 System Blocks

5.1 Fetch/Write Communication

5.1.1 SFC 228 - RW_KACHEL - Page frame direct access

Description

This SFC allows you the direct access to the page frame area of the CPU with a size of 4kbyte. The page frame area is divided into four page frames, each with a size of 1kbyte. Setting the parameters page frame number, -offset and data width, the SFC 228 enables read and write access to an eligible page frame area.

This SFC has been developed for test purposes and for building-up proprietary communication systems and is completely at the user's disposal. Please regard that a write access to the page frame area influences a communication directly!

Parameters

Name	Declaration	Type	Description
K_NR	IN	INT	Page frame number
OFFSET	IN	INT	Page frame offset
R_W	IN	INT	Access
SIZE	IN	INT	Data width
RET_VAL	OUT	BYTE	Return value (0 = OK)
VALUE	IN_ OUT	ANY	Pointer to area of data transfer

K_NR

Page frame number

- Type the page frame no. that you want to access.
  - Value range: 0 ... 3

OFFSET

Page frame offset

- Fix here an offset within the specified page frame.
  - Value range: 0 ... 1023

R_W

Read/Write

- This parameter specifies a read res. write access.
  - 0 = read access
  - 1 = write access

SIZE

Size

- The size defines the width of the data area fixed via *K_NR* and *OFFSET*. You may choose between the values 1, 2 and 4byte.

RET_VAL (Return Value)

Byte where an error message is returned to.

VALUE

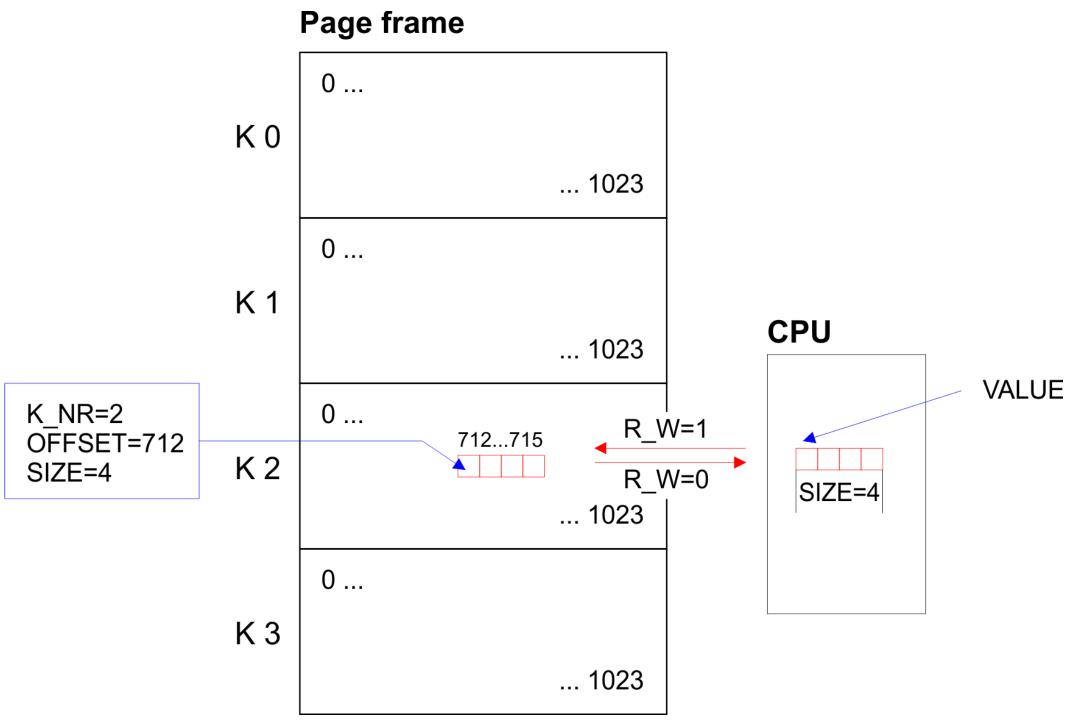
In-/output area

- This parameter fixes the in- res. output area for the data transfer.
- At a read access, this area up to 4byte width contains the data read from the page frame area.
- At a write access, the data up to 4byte width is transferred to the page frame area.
  – Parameter type: Pointer

Example

The following example shows the read access to 4byte starting with byte 712 in page frame 2. The read 4byte are stored in DB10 starting with byte 2. For this the following call is required:

```
CALL SFC 228
K_NR    :=2
OFFSET  :=712
R_W     :=0
SIZE    :=4
RET_VAL :=MB10
VALUE   :=P#DB10.DBX 2.0 Byte 4
```



Error messages

Value	Description
00h	no error
01h ... 05h	Internal error: No valid address found for a parameter
06h	defined page frame does not exist
07h	parameter SIZE ≠ 1, 2 or 4 at read access
08h	parameter SIZE ≠ 1, 2 or 4 at write access
09h	parameter R_W is ≠ 0 or 1

5.1.2 SFC 230 ... 238 - Page frame communication

5.1.2.1 Parameter description

Overview

The handling blocks allow the deployment of communication processors in the CPUs from VIPA. The handling blocks control the complete data transfer between CPU and the CPs. Advantages of the handling blocks:

- you loose only few memory space for user application
- short runtimes of the blocks

The handling blocks don't need:

- bit memory area
- time areas
- counter areas

All handling blocks described in the following use an identical interface to the user application with these parameters:

SSNR	- Interface number
ANR	- Order number
ANZW	- Indicator word (double word)
IND	- Indirect fixing of the relative start address of the data source res. destination
QANF/ZANF	- Relative start address within the type
PAFE	- Parameterization error
BLGR	- Block size

SSNR

Interface number

- Number of the logical interface (page frame address) to which the according order refers to.
  – Parameter type: Integer
  – Convenient range: 0 ... 255

ANR

Job number

- The called job number for the logical interface.
  – Parameter type: Integer
  – Convenient range: 1 ... 223

ANZW

Indicator word (double word)

- Address of the indicator double word in the user memory where the processing of the order specified under ANR is shown.
  – Parameter type: Double word
  – Convenient range: DW or MW; use either DW and DW+1 or MW and MW+2
    The value DW refers to the data block opened before the incoming call or to the directly specified DB.

IND

Kind of parameterization (direct, indirect)

- This parameter defines the kind of data on which the pointer *QANF* points.
  - 0: *QANF* points directly to the initial data of the source res. destination data.
  - 1: the pointer *QANF/ZANF* points to a memory cell, from where on the source res. destination data are defined (indirect).
  - 2: the pointer *QANF/ZANF* points to a memory area where the source res. desti-nation information lies (indirect).
  - 5: the pointer *QANF/ZANF* points to a memory cell, from where on the source res. destination data and parameters of the indicator word are defined (indirect).
  - 6: the pointer *QANF/ZANF* points to a memory area where the source res. desti-nation data and parameters of the indicator word are laying (indirect).

  - Parameter type: Integer
  - Convenient entries: 0, 1, 2, 5, 6

Please regard, that at IND = 5 res. IND = 6, the parameter ANZW is ignored!

QANF/ZANF

Relative start address of the data source res. destination and at *IND* = 5 res. *IND* = 6 of the indicator word.

- This parameter of the type "pointer" (Any-Pointer) allows you fix the relative starting address and the type of the data source (at SEND) res. the data destination (at RECEIVE).
- At *IND* = 5 res. *IND* = 6 the parameters of the indicator word are also in the data source.
  - Parameter type: Pointer
  - Convenient range: DB, M, A, E

Example:

```
P#DB10.DBX0.0 BYTE 16
P#M0.0 BYTE 10
P#E 0.0 BYTE 8
P#A 0.0 BYTE 10
```

BLGR

Block size

- During the boot process the stations agree about the block size (size of the data blocks) by means of SYNCHRON.
- A high block size = high data throughput but longer run-times and higher cycle load.
- A small block size = lower data throughput but shorter run-times of the blocks.

These block sizes are available:

Value	Block size	Value	Block size
0	Default (64byte)	4	128byte
1	16byte	5	256byte
2	32byte	6	512byte
3	64byte	255	512byte

- Parameter type: Integer
- Convenient range: 0 ... 255

PAFE

Error indication at parameterization defects

- This "BYTE" (output, marker) is set if the block detects a parameterization error, e.g. interface (plug-in) not detected or a non-valid parameterization of QUANF/ZANF.
  – Parameter type: Byte
  – Convenient range: AB 0 ... AB127, MB 0...MB 255

5.1.2.2 Parameter transfer

Direct/indirect parameterization

A handling block may be parameterized directly or indirectly. Only the "*PAFE*" parameter must always been set directly. When using the direct parameterization, the handling block works off the parameters given immediately with the block call. When using the indirect parameterization, the handling block gets only pointers per block parameters. These are pointing to other parameter fields (data blocks or data words). The parameters *SSNR*, *ANR*, *IND* and *BLGR* are of the type "integer", so you may parameterize them indirectly.

Example

Direct parameter transfer

```
CALL   SFC  230
        SSNR:=0
        ANR :=3
        IND :=0
        QANF:=P#A 0.0 BYTE 16
        PAFE:=MB79
        ANZW:=MD44
```

Indirect parameter transfer

```
CALL   SFC  230
        SSNR:=MW10
        ANR :=MW12
        IND :=MW14
        QANF:=P#DB10.DBX0.0 BYTE 16
        PAFE:=MB80
        ANZW:=MD48
```

Please note that you have to load the bit memory words with the corresponding values before.

5.1.2.3 Source res. destination definition

Overview

You have the possibility to set the entries for source, destination and *ANZW* directly or store it indirectly in a block to which the *QANF / ZANF* res. *ANZW* pointer points. The parameter *IND* is the switch criterion between direct and indirect parameterization.

Direct parameterization of source and destination details (IND = 0)

With *IND* = 0 you fix that the pointer *QANF / ZANF* shows directly to the source res. destination data. The following table shows the possible *QANF / ZANF* parameters at the direct parameterization:

QTYP/ZTYP	Data in DB	Data in MB	Data in OB Process image of the outputs	Data in IB Process image of the inputs
Pointer: Example:	`P#DBa.DBX b.0 BYTE CP#DB10.DBX 0.0 BYTE 8`	`P#M b.0 BYTE cP#M 5.0 BYTE 10`	`P#A b.0 BYTE cP#A 0.0 BYTE 2`	`P#E b.0 BYTE cP#E 20.0 BYTE 1`
DB, MB, AB, EB Definition	`P#DBa` "a" means the DB-No., from where the source data is fetched or where to the destination data is transferred.	`P#M` The data is stored in a MB.	`P#A` The data is stored in the output byte.	`P#E` The data is stored in the input byte.

QTYP/ZTYP	Data in DB	Data in MB	Data in OB Process image of the outputs	Data in IB Process image of the inputs
Valid range for "a"	0 ... 32767	irrelevant	irrelevant	irrelevant
Data / Marker Byte, OB, IB Definition	DB-No., where data fetch or write starts.	Bit memory byte number, where data fetch or write starts.	Output byte number, where data fetch or write starts.	Input byte number, where data fetch or write starts.
Valid range for "b"	0.0 ... 2047.0	0 ... 255	0 ... 127	0 ... 127
`BYTE c` Valid range for "c"	Length of the Source/ Destination data blocks in words. 1 ... 2048	Length of the Source/ Destination data blocks in bytes. 1 ... 255	Length of the Source/ Destination data blocks in bytes. 1 ... 128	Length of the Source/ Destination data blocks in bytes. 1 ... 128

Indirect parameterization of source and destination details (*IND* = 1 or *IND* = 2)

Indirect addressing means that *QANF / ZANF* points to a memory area where the addresses of the source res. destination areas and the indicator word are stored. In this context you may either define one area for data source, destination and indicator word (*IND* = 1) or each, data source, data destination and the indicator word, get an area of their own (*IND* = 2). The following table shows the possible *QANF / ZANF* parameters for indirect parameterization:

QTYP/ZTYP	IND = 1		IND = 2		
Definition	Indirect addressing for source **or** destination parameters. The source or destination parameters are stored in a DB. *QANF/ZANF:*		Indirect addressing for source and destination parameters. The source **and** destination parameters are stored in a DB in a sequential order. *QANF/ZANF:*		
	DW +0	Data type source	DW +0	Data type source	Description data source
	+2	DB-Nr. at type "DB", otherwise irrelevant	+2	DB-Nr. at type "DB", otherwise irrelevant	
	+4	Start address	+4	Start address	
	+6	Length in Byte	+6	Length in Byte	
			+8	Data type destination	Description data destination
			+10	DB-Nr. at type "DB", otherwise irrelevant	
			+12	Start address	
			+14	Length in Byte	
valid DB-No.	0 ... 32767		0 ... 32767		
Data word Definition	DW-No., where the stored data starts		DW-No., where the stored data starts		
Valid range	0.0 ... 2047.0		0.0 ... 2047.0		
Length Definition	Length of the DBs in byte		Length of the DBs in byte		
Valid range	8 fix		16 fix		

Indirect parameterization of source and destination details and *ANZW* (*IND* = 5 or *IND* = 6)

Indirect addressing means that *QANF / ZANF* points to a memory area where the addresses of the source res. destination areas and the indicator word are stored. In this context you may either define one area for data source, destination and indicator word (*IND* = 5) or each, data source, data destination and the indicator word, get an area of their own (*IND* = 6). The following table shows the possible *QANF / ZANF* parameters for indirect parameterization:

QTYP/ZTYP	IND = 5			IND = 6		
Definition	Indirect addressing for source or destination parameters and indicator word (*ANZW*). The source or destination parameters and *ANZW* are stored in a DB in a sequential order. *QANF/ZANF*			Indirect addressing for source and destination parameters and indicator word (*ANZW*). The source and destination parameters and *ANZW* are stored in a DB in a sequential order. *QANF/ZANF*		
	DW +0	Data type source	Description data source/ destination	DW +0	Data type source	Description data source
	+2	DB-Nr. at type "DB", otherwise irrelevant		+2	DB-Nr. at type "DB", otherwise irrelevant	
	+4	Start address		+4	Start address	
	+6	Length in Byte		+6	Length in Byte	
	+8	Data type destination	Description indicator word	+8	Data type destination	Description data destination
	+10	DB-Nr. at type "DB", otherwise irrelevant		+10	DB-Nr. at type "DB", otherwise irrelevant	
	+12	Start address		+12	Start address	
				+14	Length in Byte	
				+16	Data type source	Description indicator word
				+18	DB-Nr. at type "DB", otherwise irrelevant	
				+20	Start address	
valid DB-No.	0 ... 32767			0 ... 32767		
Data word Definition	DW-Nr., where the stored data starts			DW-Nr., where the stored data starts		
Valid range	0.0 ... 2047.0			0.0 ... 2047.0		
Length Definition	Length of the DBs in byte			Length of the DBs in byte		
Valid range	14 fix			22 fix		

5.1.2.4 Indicator word *ANZW*

Status and error reports

Status and error reports are created by the handling blocks:

- by the indicator word *ANZW* (information at order commissioning).
- by the parameter error byte *PAFE* (indication of a wrong order parameterization).

Content and structure of the indicator word *ANZW*

The "Indicator word" shows the status of a certain order on a CP. In your PLC program you should keep one indicator word for each defined order at hand. The indicator word has the following structure:

Byte	Bit 7 ... Bit 0
0	■ Bit 3 ... Bit 0: Error management CPU – 0: no error – 1 ... 5: CPU-Error – 6 ... 15: CP-Error ■ Bit 7 ... Bit 4: reserved
1	State management CPU ■ Bit 0: Handshake convenient (data exists) – 0: RECEIVE blocked – 1: RECEIVE released ■ Bit 1: order commissioning is running – 0: SEND/FETCH released – 1: SEND/FETCH blocked ■ Bit 2: Order ready without errors ■ Bit 3: Order ready with errors Data management handling block ■ Bit 4: Data receive/send is running ■ Bit 5: Data transmission active ■ Bit 6: Data fetch active ■ Bit 7: Disable/Enable data block – 1: released – 0: blocked
2 ... 3	Length word handling block

In the "length word" the handling blocks (SEND, RECEIVE) store the data that has already been transferred, i.e. received data in case of a Receive order, send data when there is a Send order. The announcement in the "length word" is always in byte and absolute.

Error management Byte 0, Bit 0 ... Bit 3

Those bits announce the error messages of the order. The error messages are only valid if the bit "Order ready with error" in the status bit is set simultaneously.

The following error messages may occur:

0 - **no error**

If the bit "Order ready with error" is set, the CP had to reinitialize the connection, e.g. after a reboot or RESET.

1 - **wrong Q/ZTYP at HTB**

The order has been parameterized with the wrong type label.

2 - **AG area not found**

The order impulse had a wrong parameterized DB-No.

3 - **AG area too small**

Q/ZANF and Q/ZLAE overwrite the range boundaries. Handling with data blocks the range boundary is defined by the block size. With flags, timers, counters etc. the range size depends on the AG.

4 - **QVZ-Error in the AG**

This error message means, that you chose a source res. destination parameter of the AG area, where there is either no block plugged in or the memory has a defect. The QVZ error message can only occur with the type Q/ZTYP AS, PB, QB or memory defects.

5 - **Error at indicator word**

The parameterized indicator word cannot be handled. This error occurs, if *ANZW* declared a data word res. double word, that is not (any more) in the specified data block, i.e. DB is too small or doesn’t exist.

6 - **no valid ORG-Format**

The data destination res. source isn’t declared, neither at the handling block (Q/TYP="NN") nor at the coupler block.

7 - **Reserved**

8 - **no available transfer connections**

The capacity for transfer connections is at limit. Delete unnecessary connections.

9 - **Remote error**

There was an error at the communication partner during a READ/WRITE-order.

A - **Connection error**

The connection is not (yet) established. The message disappears as soon as the connection is stable. If all connections are interrupted, please check the block itself and the bus cable. Another possibility for the occurrence of this error is a wrong parameterization, like e.g. inconsistent addressing.

B - **Handshake error**

This could be a system error or the size of the data blocks has been defined out of range.

C - **Initial error**

The wrong handling block tried to initialize the order or the size of the given data block was too large.

D - **Cancel after RESET**

This is a normal system message. With PRIO 1 and 2 the connection is interrupted but will be established again, as soon as the communication partner is online. PRIO 3 connections are deleted, but can be initialized again.

E - **Order with basic load function**

This is a normal system message. This order is a READ/WRITEPASSIV and can not be started from the AG.

F - **Order not found**The called order is not parameterized on the CP. This error may occur when the SSNR/A-No. combination in the handling block is wrong or no connection block is entered.

The bits 4 to 7 of byte 2 are reserved for extensions.

Status management Byte 1, Bit 0 ... Bit 3

Here you may see if an order has already been started, if an error occurred or if this order is blocked, e.g. a virtual connection doesn't exist any longer.

- **Bit 0 - Handshake convenient**
  - Set:
    Per plug-in according to the "delete"-announcement in the order status bit: Handshake convenient (= 1) is used at the RECEIVE block (telegram exists at PRIO 1 or RECEIVE impulse is possible at PRIO 2/3)
  - Analyze:
    Per RECEIVE block: The RECEIVE initializes the handshake with the CP only if this bit is set. Per application: for RECEIVE request (request a telegram at PRIO 1).
- **Bit 1 - Order is running**
  - Set:
    Per plug-in: when the CP received the order.
  - Delete:
    Per plug-in: when an order has been commissioned (e.g. receipt received).
  - Analyze:
    Per handling blocks: A new order is only send, when the order before is completely commissioned. Per user: when you want to know, if triggering a new order is convenient.
- **Bit 2 - Order ready without errors**
  - Set:
    Per plug-in: when the according order has been commissioned without errors.
  - Delete:
    Per plug-in: when the according order is triggered for a second time.
  - Analyze:
    Per user: to proof that the order has been commissioned without errors.
- **Bit 3 - Order ready with errors**
  - Set:
    Per plug-in: when the according order has been commissioned with errors. Error causes are to find encrypted in the high-part of the indicator word.
  - Delete:
    Per plug-in: when the according order is triggered for a second time.
  - Analyze:
    Per user: to proof that the order has been commissioned with errors. If set, the error causes are to find in the highbyte of the indicator word.

Data management Byte 1, Bit 4 ... Bit 7

Here you may check if the data transfer is still running or if the data fetch res. transmission is already finished. By means of the bit "Enable/Disable" you may block the data transfer for this order (Disable = 1; Enable = 0).

- **Bit 4 - Data fetch / Data transmission is active**
  - Set:
    Per handling block SEND or RECEIVE, if the fetch/transmission has been started, e.g. when data is transferred with the ALL-function (DMA-replacement), but the impulse came per SEND-DIRECT.
  - Delete:
    Per handling blocks SEND or RECEIVE, if the data transfer of an order is finished (last data block has been transferred).
  - Analyze:
    Per user: During the data transfer CP <<->> AG the user must not change the record set of an order. This is uncritical with PRIO 0/1 orders, because here the data transfer is realizable in one block cycle. Larger data amounts however are transferred in blocks during more AG cycles. To ensure data consistency you should proof that the data block isn’t in transfer any more before you change the content!
- **Bit 5 - Data transmission is active**
  - Set:
    Per handling block SEND, when the data transition for an order is ready.
  - Delete:
    Per handling block SEND, when the data transfer for a new order has been started (new trigger). Per user: When analysis is ready (flank creation).
  - Analyze:
    Per user: Here you may ascertain, if the record set of an order has already been transferred to the CP res. at which time a new record set concerning a running order (e.g. cyclic transition) may be started.
- **Bit 6 - Data fetch active**
  - Set:
    Per RECEIVE, when data fetch for a new order has been finished.
  - Delete:
    Per RECEIVE, when data transfer to AG for a new order (new trigger) has been started. Per user, when analyzing (edge creation).
  - Analyze:
    Per user: Here you may ascertain, if the record set of an order has already been transferred to the CP res. at what time a new record set for the current order has been transferred to the AG.
- **Bit 7 - Disable/Enable data block**
  - Set:
    Per user: to avoid overwriting an area by the RECEIVE block res. data transition of an area by the SEND block (only for the first data block).
  - Delete:
    Per user: to release the according data area.
  - Analyze:
    Per handling blocks SEND and RECEIVE: if Bit 7 is set, there is no data transfer anymore, but the blocks announce an error to the CP.

Length word Byte 2 and Byte 3

In the length word the handling blocks (SEND, RECEIVE) store the already transferred data of the current order, i.e. the received data amount for receiving orders, the sent data amount for sending orders.

Describe: - Per SEND, RECEIVE during the data transfer. The length word is calculated from: current transfer amount + amount of already transferred data

Delete: - Per overwrite res. with every new SEND, RECEIVE, FETCH. If the bit "order ready without error" res. "Data fetch/data transition ready" is set, the "Length word" contains the current source res. destination length. If the bit "order ready with error" is set, the length word contains the data amount transferred before the failure occurred.

Status and error reports

The following section lists important status and error messages of the CPU that can appear in the "Indicator word". The representation is in "HEX" patterns. The literal X means "not declared" res. "irrelevant"; No. is the error number.

X F X A - The error index "F" shows, that the according order is not defined on the CP. The state index "A" causes a block of this order (for SEND/FETCH and RECEIVE).

X A X A - The error index "A" shows that the connection of the communication order is not (yet) established. Together with the state index "A" SEND, RECEIVE and FETCH are blocked.

X 0 X 8 - The connection has been established again (e.g. after a CP reboot), the SEND order is released (SEND-communication order).

X 0 X 9 - The connection has been established again, the RECEIVE order is released (RECEIVE-communication order).

X 0 2 4 - SEND has been worked off without errors, the data was transferred.

X 0 4 5 - RECEIVE was successful, the data arrived at the AG.

X 0 X 2 - The SEND-, RECEIVE-, READ- res. WRITE order is still running. At SEND the partner is not yet ready for RECEIVE or vice versa.

Important indicator word states

Messages at SEND

State at H1	**Prio 0/1**	**Prio 2**	**Prio 3/4**
State at TCP/IP	**Prio 1**	**Prio 2**	**Prio 3**
after reboot	0 A 0 A	0 A 0 A	0 0 0 8
after connection start	X 0 X 8	X 0 X 8	.....
after initial impulse	X 0 X 2	X 0 X 2	X 0 X 2
ready without error	X 0 2 4	X 0 2 4	X 0 2 4
ready with error	X No X 8	X No X 8	X No X 8
after RESET	X D X A	X D X A	X D X 8

Messages at RECEIVE

State at H1	Prio 0/1	Prio 2	Prio 3/4
State at TCP/IP	**Prio 1**	**Prio 2**	**Prio 3**
after reboot	0 A 0 A	0 A 0 A	0 0 0 1
after connection start	X 0 X 4	X 0 0 9	.....
after initial impulse	X 0 X 2	X 0 X 2	X 0 X 2
Telegram here	X 0 X 1	.....	.....
ready without error	X 0 4 1	X 0 4 5	X 0 4 5
ready with error	X No X 8	X No X 9	X No X 9
after RESET	X D X A	X D X A	X D X 9

Messages at READ/WRITE-ACTIVE

State at H1	Prio 0/1	Prio 2	Prio 3/4
State at TCP/IP	**Prio 1**	**Prio 2**	**Prio 3**
after reboot		0 A 0 A	
after connection start		X 0 0 8	
after initial impulse		X 0 X 2	
READ ready		X 0 4 4	
WRITE ready		X 0 2 4	
ready with error		X No X 8	
after RESET		X D X A	

5.1.2.5 Parameterization error *PAFE*

The parameterization error byte *PAFE* is set (output or bit memory), when the block detects a "parameterization error", e.g. there is no interface or there is an invalid parameterization of *QANF* / *ZANF*. *PAFE* has the following structure:

Byte	Bit 7 ... Bit 0
0	■ Bit 0: error – 0: no error – 1: error, error-No. in Bit 4 ... Bit 7 ■ Bit 3 ... Bit 1: reserved ■ Bit 7 ... Bit 4: error number – 0: no error – 1: wrong ORG-Format – 2: area not found (DB not found) – 3: area too small – 4: QVZ-error – 5: wrong indicator word – 6: no Source-/Destination parameters at SEND/RECEIVE ALL – 7: interface not found – 8: interface not specified – 9: interface overflow – A: reserved – B: invalid order-No. – C: interface of CP doesn’t quit or is negative – D: Parameter *BLGR* not allowed – E: reserved – F: reserved

5.1.3 SFC 230 - SEND - Send to page frame

Description

The SEND block initializes a send order to a CP. Normally SEND is called in the cyclic part of the user application program. Although the insertion of this block into the interrupt or the time-alarm program part is possible, the indicator word (*ANZW*), however, may not be updated cyclically. This should be taken over by a CONTROL block.

The connection initialization with the CP for data transmission and for activating a SEND impulse is only started, if:

- the FB RLO (result of operation) received "1".
- the CP released the order.
  (Bit "order active" in *ANZW* = 0).

During block stand-by, only the indicator word is updated.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
ANR	IN	INT	Job number
IND	IN	INT	Mode of addressing
QANF	IN	ANY	Pointer to data source
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

SEND_ALL for data transmission

If the CP is able to take over the data directly, the SEND block transfers the requested data in one session. If the CP requests only the order parameters or the amount of the depending data is too large, the CP only gets the sending parameters res. the parameter with the first data block. The according data res. the assigned serial blocks for this order are requested from the CP by SEND_ALL to the CPU. For this it is necessary that the block SEND_ALL is called minimum one time per cycle. The user interface is for all initialization types equal, only the transfer time of the data is postponed for minimum one CPU cycle.

5.1.4 SFC 231 - RECEIVE - Receive from page frame

Description

The RECEIVE block receives data from a CP. Normally the RECEIVE block is called in the cyclic part of the user application program. Although the insertion of this block into the interrupt or the waking program part is possible, the indicator word cannot be updated cyclically. This should be taken over by a CONTROL block.

The handshake with the CP (order initialization) and for activating a RECEIVE block is only started, if

- the FB RLO received "1".
- the CP released the order (Bit "Handshake convenient" = 1).

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
ANR	IN	INT	Job number
IND	IN	INT	Mode of addressing
ZANF	IN	ANY	Pointer to data destination
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

If the block runs in stand-by only the indicator word is updated. The RECEIVE block reacts different depending from the kind of supply and the CP reaction:

- If the CP transmits a set of parameters although the RECEIVE block itself got destination parameters, the parameter set of the block has the priority above those of the CP.
- Large amounts of data can only be transmitted in blocks. Therefore you have to transmit the assigned serial blocks by means of RECEIVE_ALL to the CPU. It is necessary that the block RECEIVE_ALL is called minimum one time per application cycle and CP interface, if you want to transmit larger data amounts. You also have to integrate the RECEIVE_ALL cyclically, if the CP only uses the RECEIVE for releasing a receipt telegram and the data is transmitted via the background communication of the CPU.

5.1.5 SFC 232 - FETCH - Fetch from page frame

Description

The FETCH block initializes a FETCH order in the partner station. The FETCH order defines data source and destination and the data source is transmitted to the partner station. The CPU from VIPA realizes the definition of source and destination via a pointer parameter. The partner station provides the Source data and transmits them via SEND_ALL back to the requesting station. Via RECEIVE_ALL the data is received and is stored in Destination. The update of the indicator word takes place via FETCH res. CONTROL.

The handshake for initializing FETCH is only started, if

- the FB RLO receives "1".
- the function has been released in the according CP indicator word (order active = 0).

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
ANR	IN	INT	Job number
IND	IN	INT	Mode of addressing
ZANF	IN	ANY	Pointer to data destination
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

Information for indirect parameterization ↳ Chapter 5.1.2.3 'Source res. destination definition' on page 15

5.1.6 SFC 233 - CONTROL - Control page frame

Description

The purpose of the CONTROL block is the following:

- Update of the indicator word
- Query if a certain order of the CP is currently "active", e.g. request for a receipt telegram
- Query the CP which order is recently in commission

The CONTROL block is not responsible for the handshake with the CP, it just transfers the announcements in the order status to the parameterized indicator word. The block is independent from the RLO and should be called from the cyclic part of the application.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
ANR	IN	INT	Job number
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

ANR

If *ANR* $\neq$ 0, the indicator word is built up and handled equal to all other handling blocks. If the parameter *ANR* gets 0, the CONTROL command transmits the content of the order state cell 0 to the LOW part of the indicator words. The order state cell 0 contains the number of the order that is in commission, e.g. the order number of a telegram (set by the CP).

5.1.7 SFC 234 - RESET - Reset page frame

Description

The RESET ALL function is called via the order number 0. This resets all orders of this logical interface, e.g. deletes all order data and interrupts all active orders. With a direct function (*ANR* ≠ 0) only the specified order will be reset on the logical interface. The block depends on the RLO and may be called from cyclic, time or alarm controlled program parts.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
ANR	IN	INT	Job number
PAFE	OUT	BYTE	Parameterization error

Operating modes

The block has two different operating modes:

- RESET ALL
- RESET DIRECT

5.1.8 SFC 235 - SYNCHRON - Synchronization page frame

Description

The SYNCHRON block initializes the synchronization between CPU and CP during the boot process. For this it has to be called from the starting OBs. Simultaneously the transition area of the interface is deleted and predefined and the CP and the CPU agree about the block size.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
BLGR	IN	INT	Block size
PAFE	OUT	BYTE	Parameterization error

Block size

To avoid long cycle run-times it is convenient to split large data amounts into smaller blocks for transmitting them between CP and CPU. You declare the size of these blocks by means of "block size". A large block size = high data throughput, but also longer run-times and therefore a high cycle time strain. A small block size = smaller data throughput, but also shorter run-times of the blocks. Following block sizes are available:

Value	Block size	Value	Block size
0	Default (64byte)	4	128byte
1	16byte	5	256byte
2	32byte	6	512byte
3	64byte	255	512byte

Parameter type:	Integer
Valid range:	0 ... 255

5.1.9 SFC 236 - SEND_ALL - Send all to page frame

Description

Via the SEND_ALL block, the data is transmitted from the CPU to the CP by using the declared block size. Location and size of the data area that is to transmit with SEND_ALL, must be declared before by calling SEND res. FETCH. In the indicator word that is assigned to the concerned order, the bit "Enable/Disable" is set, "Data transmission starts" and "Data transmission running" is calculated or altered.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

ANZW

In the indicator word of the block, that is parameterized in the SEND_ALL block, the current order number is stored (0 means stand-by). The amount of the transmitted data for one order is shown in the data word of SEND_ALL which follows the indicator word.

In the following cases, the SEND_ALL command has to be called for minimum one time per cycle of the block OB 1:

- *if the CP is able to request data from the CPU independently.*
- *if a CP order is initialized via SEND, but the CP still has to request the background communication data of the CPU for this order.*
- *if the amount of data, that should be transmitted by this SEND to the CP, is higher than the declared block size.*

5.1.10 SFC 237 - RECEIVE_ALL - Receive all from page frame

Description

Via the RECEIVE_ALL block, the data received from the CP is transmitted from the CP to the CPU by using the declared block size. Location and size of the data area that is to transmit with RECEIVE_ALL, must be declared before by calling RECEIVE. In the indicator word that is assigned to the concerned order, the bit "Enable/Disable" is set, "Data transition starts" and "Data transition/fetch running" is analyzed or altered. The receiving amount is shown in the following word.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

ANZW

In the indicator word of the block, that is parameterized in the RECEIVE_ALL block, the current order number is stored. In the stand-by running mode of RECEIVE_ALL the block indicator word is deleted.

In the following cases, the RECEIVE_ALL command has to be called for minimum one time per cycle of the block OB 1:

– *if the CP should send data to the CPU independently.*
– *if a CP order is initialized via RECEIVE, but the CP still has to request the "background communication" data of the CPU for this order.*
– *if the amount of data that should be transmitted to the CPU by this RECEIVE, is higher than the declared block size.*

5.1.11 SFC 238 - CTRL1 - Control1 page frame

Description

This block is identical to the CONTROL block SFC 233 except that the indicator word is of the type Pointer and that it additionally includes the parameter *IND*, reserved for further extensions. The purpose of the CONTROL block is the following:

- Update of the indicator word.
- Query if a certain order of the CP is currently active, e.g. request for a receipt telegram
- Query the CP which order is recently in commission

The CONTROL block is not responsible for the handshake with the CP; it just transfers the announcements in the order status to the parameterized indicator word. The block is independent from the RLO and should be called from the cyclic part of the application.

Parameters

Name	Declaration	Type	Description
SSNR	IN	INT	Interface number
ANR	IN	INT	Job number
IND	IN	INT	Reserved
PAFE	OUT	BYTE	Parameterization error
ANZW	IN_OUT	DWORD	Indicator word

ANR

If *ANR* ≠ 0, the indicator word is built up and handled equal to all other handling blocks. If the parameter *ANR* gets 0, the CONTROL command transmits the content of the order state cell 0 to the LOW part of the indicator words. The order state cell 0 contains the number of the order that is in commission, e.g. the order number of a telegram (set by the CP).

IND

The parameter *IND* has no functionality at this time and is reserved for further extensions.

ANZW

The indicator word *ANZW* is of the type Pointer. This allows you to store the indicator word in a data block.

5.2 MMC Functions standard CPUs

5.2.1 SFC 220 ... 222 - MMC Access

Overview

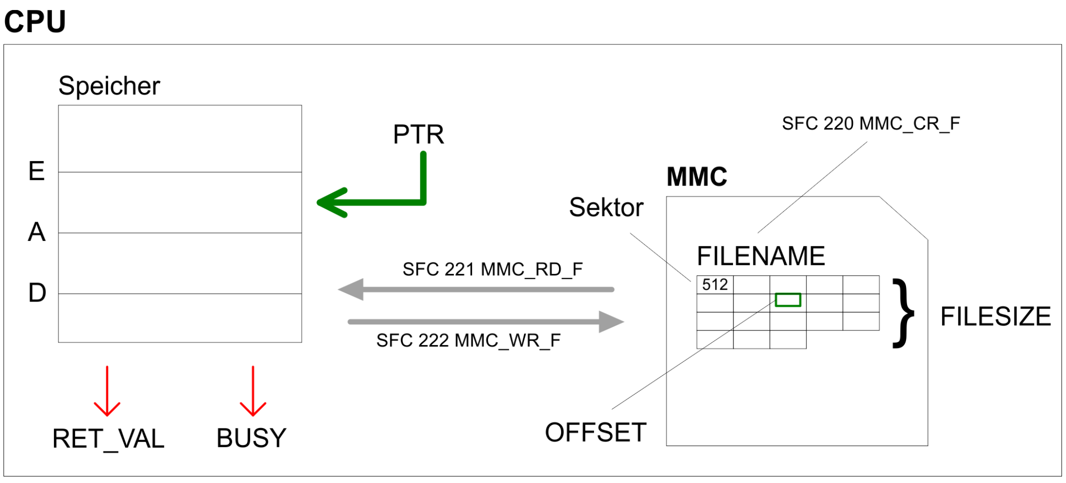
By means of these blocks there is the possibility to integrate MMC access to your application program. Here a new file may be created respectively an existing file may be opened for accessed when a MMC is plugged-in. As long as you do not open another file, you may access this file via read/write commands.

Restrictions

For deploying the SFCs 220, 221 and 222, you have to regard the following restrictions:

- A read res. write access to the MMC is only possible after creation res. opening of the file via SFC 220.
- The data on MMC must not be fragmented, for only complete data blocks may be read res. written.
- When transferring data to the MMC from an external reading device, they may be fragmented, i.e. the data is divided into blocks. This may be avoided by formatting the MMC before the write access.
- At a write access from the CPU to the MMC, the data is always stored not fragmented.
- When opening an already existing file, you have to use the same *FILENAME* and *FILESIZE* that you used at creation of this file.
- A MMC is structured into sectors. Every sector has a size of 512byte. Sector overlapping writing or reading is not possible. Access to sector overlapping data is only possible by using a write res. read command for every sector. By giving the offset, you define the according sector.

The following picture shows the usage of the single SFCs and their variables:



For read and write accesses to the MMC, you firstly have to open the file with SFC 220!

5.2.2 SFC 220 - MMC_CR_F - create or open MMC file

Overview

By means of this SFC a new file may be created respectively an existing file may be opened for accessed when a MMC is plugged-in. As long as you do not open another file, you may access this file via read/write commands. For more detailed information to this and to the restrictions *Chapter 5.2.1 'SFC 220 ... 222 - MMC Access' on page 34.*

Since calling the SFC from the OB 1 can result in a cycle time-out, instead of this you should call the SFC from the OB 100.

Parameters

Name	Declaration	Type	Description
FILENAME	IN	STRING[254]	Name of file
FILESIZE	IN	DWORD	Size of file
RET_VAL	OUT	WORD	Return value (0 = OK)

FILENAME

Type in the file name used to store the data on the MMC. The name inclusive end ID may not exceed a maximum length of 13 characters:

- 8 characters for name
- 1 character for "."
- 3 characters for file extension
- 1 character 00h as end ID

For software technical reasons you have to enter 00h into the byte next to the file name (end ID of the file name).

FILESIZE

The *FILESIZE* defines the size of the user data in byte. When accessing an already existing file, it is mandatory to give not only the *FILENAME* but also the *FILESIZE*. The entry of a "Joker" length is not supported at this time.

Structure

Byte 0	Byte 1	Byte 2	Byte 3	...	Byte 255
Max. length	occupied length	ASCII value 1	ASCII value 2	...	ASCII value 254

RET_VAL (Return Value)

Word that returns a diagnostic/error message. 0 means OK.

Value	Description
Diagnostic messages	
0000h	No errors (appears if new file is generated).
0001h	File already exists, is not fragmented and the length value is identical or smaller.
8001h	No or unknown type of MMC is plugged-in.
Error messages	
8002h	No FAT on MMC found.
A001h	File name missing. This message appears if file name is inside a not loaded DB.

Value	Description
A002h	File name wrong (not 8.3 or empty)
A003h	File exists but *FILESIZE* too bigger than existing file.
A004h	File exists but is fragmented and cannot be opened.
A005h	Not enough space on MMC.
A006h	No free entry in root directory. Depending on the used MMC there may be min. 16 up to max. 512 entries in the root directory.
B000h	An internal error occurred.

5.2.3 SFC 221 - MMC_RD_F - read from MMC file

Description

Via the SFC 221 you may read data from a MMC. For read and write accesses to the MMC, you firstly have to open the file with SFC 220 and it has to be not fragmentized. For more detailed information to this and to the restrictions ✎ *Chapter 5.2.1 'SFC 220 ... 222 - MMC Access' on page 34*.

Parameters

Name	Declaration	Type	Description
PTR	IN	ANY	Pointer to area for reading data
OFFSET	IN	DWORD	Offset of data within the file
BUSY	OUT	BOOL	Job state
RET_VAL	OUT	WORD	Return value (0 = OK)

PTR

This variable of the type pointer points to a data area in the CPU where the content of the MMC has to be written to.

OFFSET

Here you define the start address inside the file on the MMC from where on the data has to be transferred to the CPU.

BUSY

During data transfer this bit remains set. The bit is reset as soon as the data transfer is complete.

RET_VAL (Return Value)

Word that returns a diagnostic/error message. 0 means OK.

Value	Description
0000h	No errors (data was read)
8001h	No or unknown type of MMC is plugged-in
8002h	No FAT found on MMC
9000h	Bit reading has been tried (Boolean variable). Bit reading is not possible.
9001h	Pointer value is wrong (e.g. points outside DB)

Value	Description
9002h	File length exceeded
9003h	Sector limit of 512 has been tried to overrun. Sector overrun reading is not possible.
B000h	An internal error occurred.

5.2.4 SFC 222 - MMC_WR_F - write to MMC file

Description

Via the SFC 222, you may write to the MMC. For read and write accesses to the MMC, you firstly have to open the file with SFC 220 and it has to be not fragmentized. For more detailed information to this and to the restrictions ✎ *Chapter 5.2.1 ‘SFC 220 ... 222 - MMC Access’ on page 34*.

Parameters

Name	Declaration	Type	Description
PTR	IN	ANY	Pointer to area for writing data
OFFSET	IN	DWORD	Offset of data within the file
BUSY	OUT	BOOL	Job state
RET_VAL	OUT	WORD	Return value (0 = OK)

PTR

This variable of the type pointer points to a data area from where on the data starts that will be written to the MMC.

OFFSET

This defines the beginning of the data inside the file on the MMC where the data is written to.

BUSY

During data transfer this Bit remains set. The Bit is reset as soon as the data transfer is complete.

RET_VAL (Return Value)

Word that returns a diagnostic/error message. 0 means OK.

Value	Description
0000h	No errors
8001h	No or unknown type of MMC is plugged-in.
8002h	No FAT found on MMC.
9000h	Bit writing has been tried (Boolean variable). Bit writing is not possible.
9001h	Pointer value is wrong (e.g. points outside DB).
9002h	File length exceeded.
9003h	Sector limit of 512 has been tried to overrun. Sector overrun reading is not possible.
B000h	An internal error occurred.

5.3 File Functions SPEED7 CPUs

5.3.1 FC/SFC 195 and FC/SFC 208...215 - Memory card access

Overview

The FC/SFC 195 and FC/SFC 208 ... FC/SFC 215 allow you to include the memory card access into your user application. The following parameters are necessary for the usage of the FC/SFCs:

HANDLE, FILENAME

The access takes place via a *HANDLE* number. That is assigned to a *FILENAME* via a call of the FC/SFC 208 FILE_OPN res. FC/SFC 209 FILE_CRE. At the same time a max. of 4 *HANDLE* may be opened (0 ... 3). To close an opened file call the FC/SFC 210 FILE_CLO and thus release the *HANDLE* again.

MEDIA

As media format set 0 for the MMC. Other formats are not supported at this time.

ORIGIN, OFFSET

Read and write start with the position of a write/read flag. After opening res. creation of a file, the write/read flag is at position 0. With FC/SFC 213 FILE_SEK you may shift the write/read flag from an *ORIGIN* position for an *OFFSET* (number Bytes).

REQ, BUSY

- With *REQ* = 1 you activate the according function.
- *REQ* = 0 returns the current state of a function via *RETVAL*.
- *BUSY* = 1 monitors that the according function is in process.

RETVAL

After the execution of a function *RETVAL* returns a number code:

RETVAL = 0:	Function has been executed without errors.
0 < RETVAL < 7000h:	*RETVAL* = Length of the transferred data (only FC/SFC 211 and FC/SFC 212).
7000h ≤ RETVAL < 8000h:	Monitors the execution state of the function.
RETVAL ≥ 8000h:	Indicates an error that is described more detailed in the according FC/SFC.

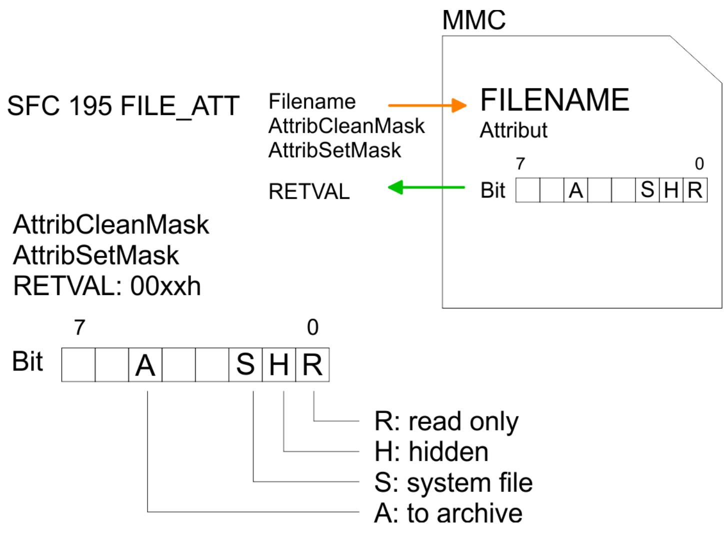
CAUTION!
For the access of the memory card you must regard the following hints. Nonobservance may cause data loss at the memory card:

– A max. of 4 Handle (0 ... 3) may be used at the same time!
– File names must follow the 8.3 format or special character!
– These FC/SFCs only gives you access to the top directory level (Root directory) of the memory card!
– You may only rename or delete files that you've closed before with FC/SFCs 210 FILE_CLO!

5.3.2 FC/SFC 195 - FILE_ATT - Change file attributes

Description

In the root directory of the memory card the file attributes may be changed by FILE_ATT. Here enter a file name. The corresponding attributes may be reset with *ATTRIBCLEAN-MASK* respectively set with *ATTRIBSETMASK* by given bit pattern. Setting takes priority over resetting. After job execution the current state of the attributes is returned with *RETVAL* 00xxh. For determination of the current file attributes by *RETVAL*, the parameters *ATTRIBCLEANMASK* and *ATTRIBSETMASK* may be set to value 00h.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
MEDIA	IN	INT	0 = MMC
FILENAME	IN	STRING[254]	Name of file (must be in 8.3 format)
ATTRIBCLEANMASK	IN	BYTE	Bit pattern of attributes to clean
ATTRIBSETMASK	IN	BYTE	Bit pattern of attributes to set
RETVAL	OUT	WORD	Return value (00xxh=OK with xx: attributes)
BUSY	OUT	BOOL	Function is busy

RETVAL (Return value)

Return codes of *RETVAL*:

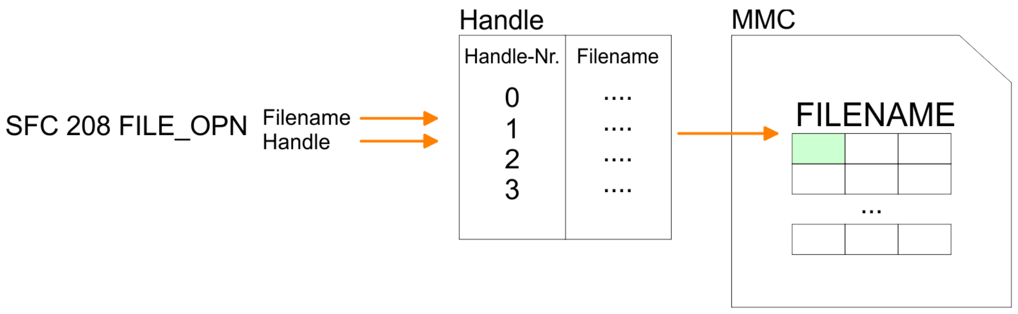
Code	Description
00xxh	OK, attributes have been changed with xx: attributes
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
A001h	The defined *MEDIA* type is not valid
A002h	Error in parameter *ATTRIBSETMASK*

Code	Description
A004h	File *FILENAME* is not found
A005h	*FILENAME* is a directory
A006h	File is just open
A007h	Memory card is write protected
A010h	File error FILENAME
A100h	General file system error (e.g. no memory card plugged)

5.3.3 FC/SFC 208 - FILE_OPN - Open file

Description

You may open a file on the memory card with FC/SFC 208. Here a *HANDLE* is connected to a *FILENAME*. By using the *HANDLE* you now have read and write access to the file until you close the file again with the FC/SFC 210 FILE_CLO. *REQ* = 1 initializes the function. After the opening the read/write flag is at 0.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
MEDIA	IN	INT	0 = MMC
FILENAME	IN	STRING[254]	Name of file (must be in 8.3 format)
HANDLE	IN	INT	Index of file 0 ... 3
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy

RETVAL (Return value)

Codes that are returned by *RETVAL*:

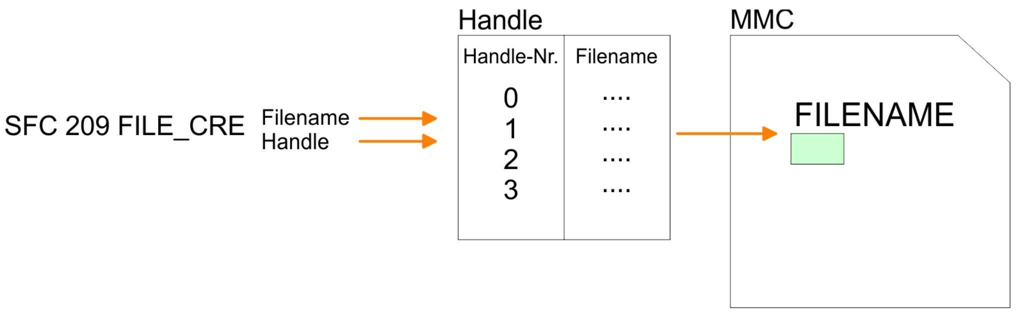
Code	Description
0000h	OK
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8010h	Parameter *FILENAME* is not present (e.g. DB not loaded).

Code	Description
8011h	Error *FILENAME* (not conform with 8.3 or special character)
8100h	The defined *HANDLE* is not valid
9001h	*HANDLE* is assigned to another file
9002h	Another function has been called via this *HANDLE* and is ready
9003h	Another function has been called via this *HANDLE* and is ready
A000h	System internal error occurred
A001h	The defined *MEDIA* type is not valid
A003h	A general error in the file system occurred
A004h	The in *FILENAME* defined file doesn't exist or is a directory
A100h	General file system error (e.g. no memory card plugged)

5.3.4 FC/SFC 209 - FILE_CRE - Create file

Description

By using this block you may create a new file with the entered file name on the memory card (if plugged) and open it for read/write access. Please regard that you may only create files at the top directory level. *REQ* = 1 initializes the function. After opening, the write /read flag is at 0.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
MEDIA	IN	INT	0 = MMC
FILENAME	IN	STRING[254]	Name of file (must be in 8.3 format)
HANDLE	IN	INT	Index of file 0 ... 3
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy

RETVAL (Return value)

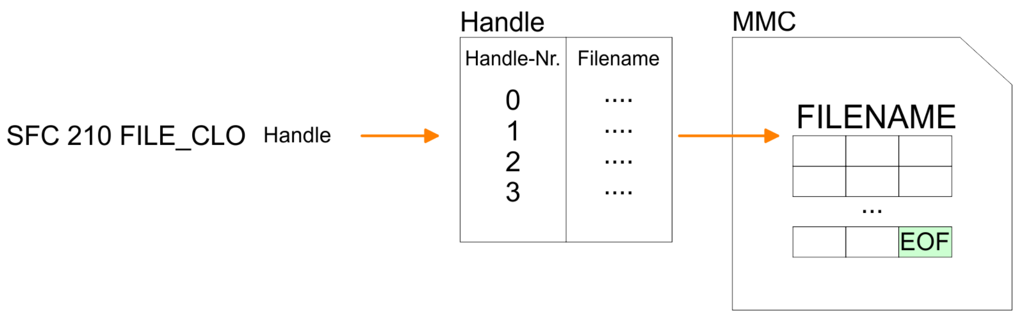
Codes that are returned by RETVAL:

Code	Description
0000h	OK
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8010h	Parameter *FILENAME* is not present (e.g. DB not loaded)
8011h	Error *FILENAME* (not conform with 8.3 or special character)
8100h	The defined HANDLE is not valid
9001h	HANDLE is assigned to another file
9002h	Another function has been called via this *HANDLE* and is ready
9003h	Another function has been called via this *HANDLE* and is not ready
A000h	System internal error occurred
A001h	The defined *MEDIA* type is not valid
A003h	A general error in the file system occurred
A004h	No root-entry is available in the directory
A005h	Memory card is write-protected
A100h	General file system error (e.g. no memory card plugged)

5.3.5 FC/SFC 210 - FILE_CLO - Close file

Description

This block allows you to close an opened file. Here an EOF (**E**nd **o**f **F**ile) is added, the file is closed and the *HANDLE* released. *REQ* = 1 initializes the function.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
HANDLE	IN	INT	Index of file 0 ... 3
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy

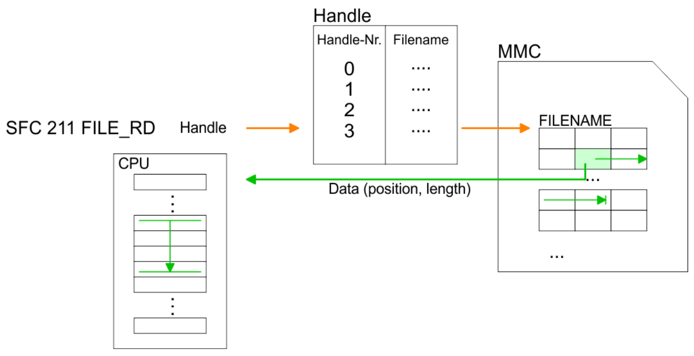
RETVAL (Return value) Codes that are returned by *RETVAL*:

Code	Description
0000h	OK
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8100h	The defined *HANDLE* is invalid
9001h	The *HANDLE* is not assigned to a file name
9002h	Another function has been called via this *HANDLE* and is ready
9003h	Another function has been called via this *HANDLE* and is not ready
A000h	System internal error occurred
A100h	General file system error (e.g. no memory card plugged)

5.3.6 FC/SFC 211 - FILE_RD - Read file

Description

This allows you to transfer data from the memory card to the CPU via the opened *HANDLE* starting from an ORIGIN position (position of the read-/write flag). During every call you may transfer a max. of 512byte. By setting of *DATA* you define storage place and length of the write area in the CPU. *REQ* = 1 initializes the function.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
HANDLE	IN	INT	Index of file 0 ... 3
DATA	IN	ANY	Pointer to PLC memory and data length
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy

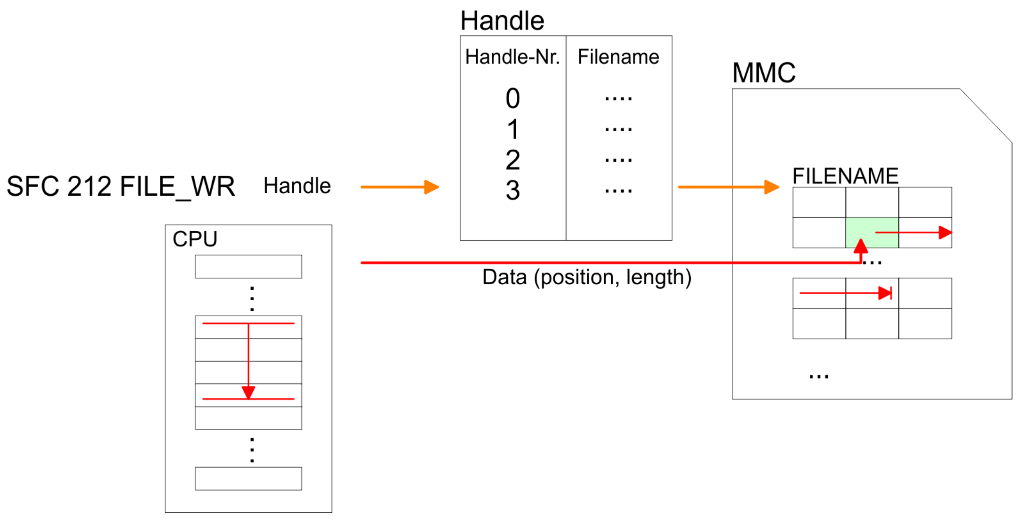
RETVAL (Return value) Codes that are returned by *RETVAL*:

Code	Description
0xxxh	0 = OK, 0xxx = Length of read data
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8010h	Pointer in *DATA* has type *BOOL*
8011h	Pointer in *DATA* cannot be decoded (e.g. DB not loaded)
8012h	Data length exceeds 512byte
8013h	A write access to a write-protected DB happened
8100h	The defined *HANDLE* is not valid
9001h	For this *HANDLE* no file is opened.
9002h	Another function has been called via this *HANDLE* and is ready
9003h	Another function has been called via this *HANDLE* and is not ready
A000h	System internal error occurred
A003h	Internal error
A100h	General file system error (e.g. no memory card plugged)

5.3.7 FC/SFC 212 - FILE_WR - Write file

Description

Use this block for write access to the memory card. This writes data from the position and length of the CPU defined under *DATA* to the memory card via the according *HANDLE* starting at the write/read position. During every call you may transfer a max. of 512byte. *REQ* = 1 initializes the function.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
HANDLE	IN	INT	Index of file 0 ... 3

Parameter	Declaration	Data type	Description
DATA	IN	ANY	Pointer to PLC memory and data length
RETVAL	OUT	WORD	Return value
BUSY	OUT	BOOL	Function is busy

The parameter *RETVAL* returns the length of the written data. The block doesn't announce an error message that the MMC is full. The user has to check himself if the number of the bytes to write corresponds to the number of written bytes returned by *RETVAL*.

RETVAL (Return value)

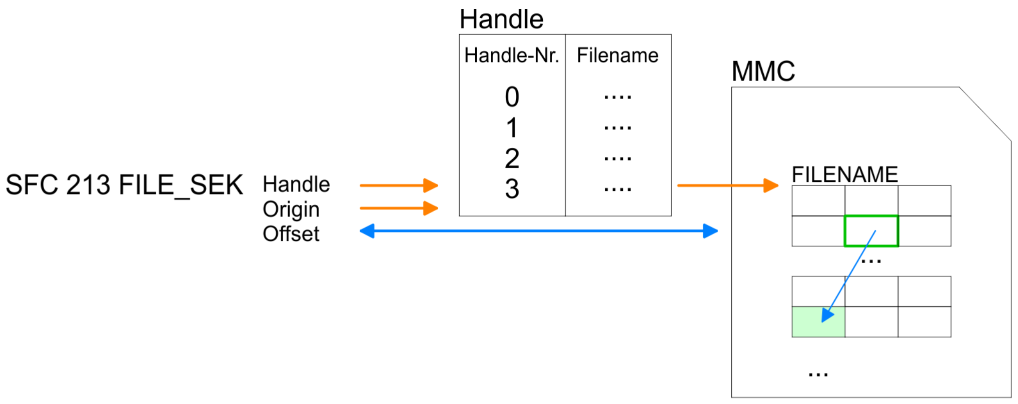
Codes that are returned by *RETVAL*:

Code	Description
0xxxh	0 = OK, 0xxx = Length of written data
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8010h	Pointer in *DATA* has type BOOL
8011h	Pointer in *DATA* cannot be decoded (e.g. DB not loaded)
8012h	Data length exceeds 512byte
8100h	The defined *HANDLE* is not valid
9001h	For this Handle no file is opened
9002h	Another function has been called via this *HANDLE* and is ready
9003h	Another function has been called via this *HANDLE* and is not ready
A000h	System internal error occurred
A002h	File is write-protected
A003h	Internal error
A004h	Memory card is write-protected
A100h	General file system error (e.g. no memory card plugged)

5.3.8 FC/SFC 213 - FILE_SEK - Position pointer

Description

FILE_SEK allows you to detect res. alter the position of the write-/read flag of the according *HANDLE*. By setting *ORIGIN* as start position and an *OFFSET* you may define the write-/read flag for the according *HANDLE*. *REQ* = 1 starts the function.



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
HANDLE	IN	INT	Index of file 0 ... 3
ORIGIN	IN	INT	0 = file start, 1 = current position, 2 = file end
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy
OFFSET	INOUT	DINT	Offset write-/read flag

RETVAL (Return value)

Codes that are returned by *RETVAL*:

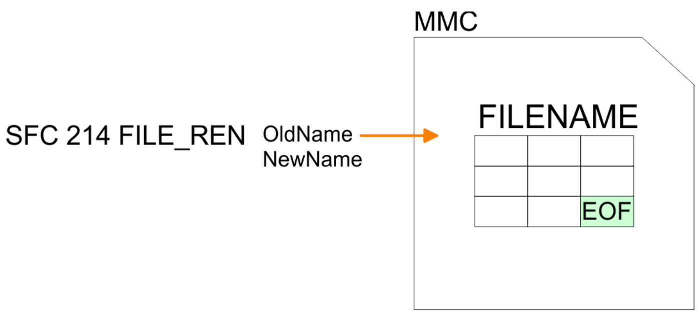
Code	Description
0000h	OK, *OFFSET* contains the current write-/read position
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8100h	The defined *HANDLE* is not valid
9001h	For this *HANDLE* no file is opened
9002h	Another function has been called via this *HANDLE* and is ready
9003h	Another function has been called via this *HANDLE* and is not ready
A000h	System internal error occurred
A004h	*ORIGIN* parameter is defective
A100h	General file system error (e.g. no memory card plugged)

5.3.9 FC/SFC 214 - FILE_REN - Rename file

Description

Using FILE_REN you may alter the file name defined in *OLDNAME* to the file name that you type in *NEWNAME*.

CAUTION!
Please regard that you may only rename files that you've closed before with FILE_CLO. Nonobservance may cause data loss at the memory card!



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
MEDIA	IN	INT	0 = MMC
OLDNAME	IN	STRING[254]	Old name of file (must be in 8.3 format)
NEWNAME	IN	STRING[254]	New name of file (must be in 8.3 format)
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy.

RETVAL (Return value)

Codes that are returned by *RETVAL*:

Code	Description
0000h	OK, file has been renamed
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8010h	Parameter *OLDNAME* is not present (e.g. DB not loaded)
8011h	Error *OLDNAME* (not conform with 8.3 format or special character)
8020h	Parameter *NEWNAME* is not present (e.g. DB not loaded)

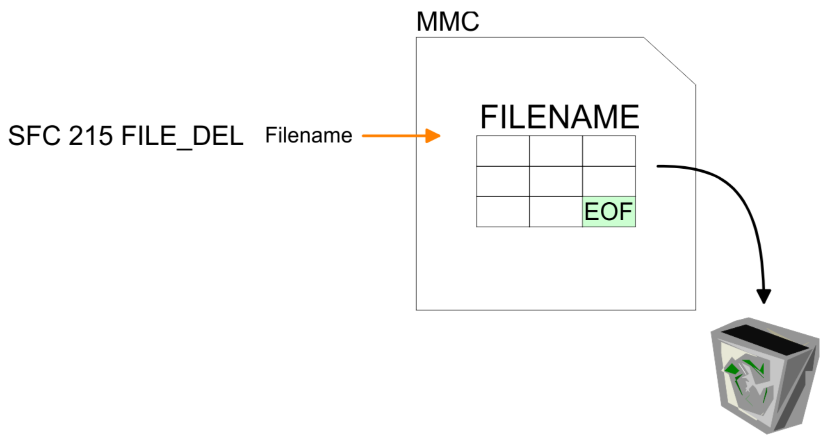
Code	Description
8021h	Error *NEWNAME* (not conform with 8.3 format or special character)
A000h	System internal error occurred
A001h	The defined MEDIA type is not valid
A003h	The new filename NEWNAME already exists
A004h	File *OLDNAME* is not found
A006h	File *OLDNAME* is just open
A007h	Memory card write-protected
A100h	Error occurs when file creation (e.g. no memory card plugged)

5.3.10 FC/SFC 215 - FILE_DEL - Delete file

Description

This block allows you to delete a file at the memory card. For this, type the file name of the file to delete under *FILENAME*.

CAUTION!
Please regard that you may only delete files that you've closed before with FILE_CLO. Nonobservance may cause data loss at the memory card!



Parameters

Parameter	Declaration	Data type	Description
REQ	IN	BOOL	Activate function
MEDIA	IN	INT	0 = MMC
FILENAME	IN	STRING[254]	Name of file (must be in 8.3 format)
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy.

RETVAL (Return value)

Codes that are returned by *RETVAL*:

Code	Description
0000h	OK, file has been deleted
7000h	*REQ* = 0, *BUSY* = 0 (nothing present)
7001h	*REQ* = 1, 1. call
7002h	Block is executed
8010h	Parameter *FILENAME* is not available (e.g. DB not loaded)
8011h	*FILENAME* is defective (e.g. is not conform with 8.3 format or special character)
A000h	System internal error occurred
A001h	The defined *MEDIA* type is not valid
A002h	The file is write-protected
A004h	File *FILENAME* is not found
A005h	*FILENAME* is a directory - you cannot delete
A006h	File is just open
A007h	Memory card is write-protected
A100h	General file system error (e.g. no memory card plugged)

5.4 System Functions

5.4.1 SFC 75 - SET_ADDR - Set PROFIBUS MAC address

Description

With this SFC you can change the MAC address of the integrated PROFIBUS interface of a CPU. The function is only possible in the passive DP slave mode. To identify the diagnostic address is used. The SFC is asynchronous and can be applied only to one interface. At STOP and subsequent warm start the set network address is retained. With PowerOFF-PowerON or on overall reset the interface gets the configured node number The DP slave consistently assumes the identity of the DP slave with the new address. For the DP master the DP slave with the old address fails and a DP slave with the new address returns. If an address is selected, which is already used by another node on the DP line, then both slaves fail in accordance to the DP communication.

Parameters

Parameter	Declaration	Data type	Memory area	Description
REQ	INPUT	BOOL	I, Q, M, D, L	Function request with *REQ* = 1
LADDR	INPUT	WORD	I, Q, M, D, L	Identification of the interface
ADDR	INPUT	BYTE	I, Q, M, D, L	New node address
RET_VAL	OUTPUT	INT	I, Q, M, D, L	Error code
BUSY	OUTPUT	BOOL	I, Q, M, D, L	*BUSY* = 1: In progress

RET_VAL (return value)

Value	Description
0000h	Job has been executed without error
7000h	Function request with *REQ* = 0 (call without processing) *BUSY* is set to 0, no data transfer is active
7001h	First call with *REQ* = 1: Data transfer started *BUSY* is set to 1
7002h	Intermediate call (*REQ* irrelevant): Data transfer started *BUSY* is set to 1
8xyyh	General error information *Chapter 4.1 'General and Specific Error Information RET_VAL' on page 8*
8090h	Identification of the interfaces: Logical address is not valid
8091h	New node address is not valid
8093h	Identification of the interfaces: Logical address is no interface
809Bh	Function not executable (e.g.. interface is no DP slave or active)
80C3h	There are no resources (e.g. multiple call of the SFC)

5.4.2 FC/SFC 193 - AI_OSZI - Oscilloscope-/FIFO function

Description

- The FC/SFC 193 serves for controlling the oscilloscope-/FIFO function of analog input channels with this functionality.
- It allows to start the recording and to read the buffered data.
- Depending upon the parameterization there are the following possibilities:

Oscilloscope operation

- Depending on the trigger condition at edge evaluation the monitoring of the configured channel may be started respectively at manual operation the recording may be started.
- The recorded measuring values may be accessed by the FC/SFC 193 as soon as the buffer is full.

FIFO operation

- Start the recording.
- Read the puffer at any time.

The FC/SFC may only be called from on level of priority e.g. only from OB 1 or OB 35.

The module is to be parameterized before.

For starting and reading in each case the FC/SFC 193 is to be called. The differentiation of both variants takes place in the parameter MODE.

Parameters

Parameter	Declaration	Data type	Function depending on MODE
REQ	IN	BOOL	Execute function (start/read)
LADR	IN	WORD	Base address of the module
MODE	IN	WORD	Mode (start/read)
CHANNEL	IN	BYTE	Channel to be read
OFFSET	IN	DWORD	Address offset for reading (not FIFO operation)
RECORD	IN	ANY	Memory for the read data
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy
TIMESTAMP	OUT	DWORD	Time stamp (only at edge evaluation)
LEN	INOUT	DWORD	Number of values to be handled per channel

REQ

- Depending on the set *MODE* when the bit is set the recording respectively the reading may be started.
- Depending on the trigger condition at edge evaluation the monitoring of the configured channel may be started respectively at manual operation the recording may be started.
- The data are read from the module, if "read" is set at *MODE*.

LADR

Logical basic address of the module.

MODE

The FC/SFC 193 may be called with 3 different modes. The corresponding mode may be set by the parameter *MODE*. The configured mode is executed by setting *REQ*. The following values are supported:

- 01h: Starts recording respectively edge monitoring depending upon the parameterization.
- 00h: Read data within several cycles until *BUSY* = 0.
- 80h: Read data with one access.

CHANNEL

Here the channel is specified to be read. With each call one channel may be read. This parameter is irrelevant at start calls with *MODE* = 01h.

OFFSET

- Offset specifies an address offset for the reading process. By this you get access to sub-ranges of the recorded data.
- The value for the maximum offset depends on the number of values, which were recorded per channel.
- *OFFSET* is not supported in FIFO operation. It will be ignored.

RECORD

- Here an area for the read values to be stored at may be defined.
- In FIFO operation every value of the selected channel may be read, which were stored up to the time of start reading.
- Please regard that the buffer has a sufficient size for the data to be buffered, otherwise an error is reported.

BUSY

- *BUSY* = 1 indicates that the function just processed.
- *BUSY* = 0 indicates that the function is finished.

TIMESTAMP

- There is an internal clock with a resolution of 1µs running in every SPEED-Bus module.
- The returned value corresponds to the time at the SPEED-Bus module, on which the trigger event occurred.
- *TIMESTAMP* is only available at the edge triggered oscilloscope operation.
- It is valid as long as the job is running (*RETVAL* = 7xxxh) and bit 4 of byte 0 is set respectively the job has been finished without an error (*RETVAL* = 0000h).

LEN

The length parameter realized as IN/OUT is variably interpreted depending on the selected mode at the function call.

Mode: start (*MODE*: = 01h)

At *MODE* = 01h this parameter may only be used at the manual oscilloscope start. Here the requested number of values per channel to be buffered may be assigned. In this mode there is no value reported by *LEN*.

Mode: read (*MODE*: = 00h or 80h)

At *MODE* = 00h respectively 80h the number of values to be read may be set. This parameter is ignored in FIFO operation. The number of the read values is returned by *LEN*.

RETVAL (Return value)

In addition to the module specific error codes listed here, there general FC/SFC error information may be returned as well.

RETVAL	Description depending on the *BUSY*-Bit	BUSY
Byte		
0	Bit 1, 0:	
	00: Call with REQ: = 0 (idle, waiting for *REQ* = 1)	0
	01: First call with *REQ*: = 1	1
	10: Subsequent call with *REQ*: = 1	1
	11: Oscilloscope is just recording	1
	Bit 2: REQ: = 1, but recording was not yet started. (*MODE*: = 00h or *MODE*: = 80h)	0
	Bit 3: reserved	-
	Bit 4: Trigger event occurred and recording is just running.	1
	Bit 5: Waiting for trigger event	1
	Bit 7…6: reserved	-
1	Bit 0: reserved	-
	Bit 1: The number of recorded values exceeds the target area defined by *RECORD* (in words).	0
	Bit 2: The number of the recorded values exceeds the area defined by *LEN* and *OFFSET*.	0
	Bit 3: Buffer overflow in FIFO operation.	0

RETVAL	Description depending on the *BUSY*-Bit	BUSY
	Bit 7...4:	
	0000: Job finished without an error	0
	0111: Job still running	1
	1000: Job finished with error	0

Job finished without an error

RETVAL	Description depending on the *BUSY*-Bit	BUSY
0000h	Job was finished without an error.	0

Job finished with error

RETVAL	Description depending on the *BUSY*-Bit	BUSY
8002h:	Oscilloscope-/FIFO function is not configured.	0
8003h:	An internal error occurred - please contact VIPA.	0
8005h:	The selected channel may not be read - wrong channel number.	0
8007h:	The value at *OFFSET* exceeds the number of recorded values.	0
8090h:	There is no SPEED-Bus module with this address available.	0
80D2h:	*LADR* exceeds the peripheral address area.	0

5.4.3 FC/SFC 194 - DP_EXCH - Data exchange with CP342S

Description

With the FC/SFC 194 you can exchange data between your CPU and a PROFIBUS DP master, which is connected via SPEED-Bus. Normally each PROFIBUS DP master embeds its I/O area into the peripheral area of the CPU. Here you can address a periphery range of 0 ... 2047 via the hardware configuration. Since this limits the maximum number of PROFIBUS DP master modules at the SPEED-Bus, there is the possibility to deactivate the mapping at the appropriate DP master and to activate instead the access via handling blocks. Here you can write data from the CPU in a defined area of the DP master and read data from a defined area of the DP master.

Parameters

Parameter	Declaration	Data type	Functionality depending on MODE
LADR	IN	WORD	Base address of the DP master module on the SPEED-Bus
MODE	IN	WORD	Modus (0 = read / 1 = write)
LEN	IN	WORD	Length of the data area in the DP master
OFFSET	IN	DWORD	Begin of the data area in the DP master
RETVAL	OUT	WORD	Return value (0 = OK)
DATA	IN OUT	ANY	Pointer to the data area of the CPU

LADR

Logical base address of the module.

MODE

Den FC/SFC 194 may be called with the following modes:

- 0000 = Transfer data from the DP master to the CPU.
- 0001 = Transfer data from the CPU to the DP master.

LEN

Here the length of the data area in the DP master is defined.

OFFSET

Here the beginning of the data area in the DP master is defined. Please consider that the area defined via *OFFSET* and *LEN* does not exceed the area defined of the DP master by the hardware configuration.

RETVAL (Return value)

In addition to the module-specific error codes listed here, as return value there are also general error codes possible for FC/SFCs . *Chapter 4.1 'General and Specific Error Information RET_VAL' on page 8*

RETVAL	Description
0000h	No error
8001h	*LADR* could not be assigned to a DP master at the SPEED-Bus.
8002h	The value of the parameter *MODE* is out of range.
8003h	The value of the parameter *LEN* is 0.
8004h	The value of the parameter *LEN* is greater than the data area defined at *DATA*.
8005h	The area defined by *OFFSET* and *LEN* is out of the range 0 …2047.

RETVAL	Description
8006h	The DP master specified by *LADR* is not configured for access via handling block. Activate in the properties of the DP master "IO-Mode HTB".
8008h	There are gap(s) in the input area.
8009h	There are gap(s) in the output area.
8010h	Error while accessing the input area (e.g. DP master is not reachable)
8011h	Error while accessing the output area (e.g. DP master is not reachable)
8Fxxh	Error at DATA (xx) ➪ *Chapter 4.1 'General and Specific Error Information RET_VAL' on page 8*

5.4.4 FC/SFC 219 - CAN_TLGR - CANopen communication

FC/SFC 219 CAN_TLGR SDO request to CAN master

Every SPEED7-CPU provides the integrated FC/SFC 219. This allows you to initialize a SDO read or write access from the PLC program to the CAN master. For this you address the master via the slot number and the destination slave via its CAN address. The process data is defined by the setting of *INDEX* and *SUBINDEX*. Via SDO per each access a max. of one data word process data can be transferred.

Parameters

Parameter	Declaration	Data type	Description
REQUEST	IN	BOOL	Activate function
SLOT_MASTER	IN	BYTE	SPEED-Bus slot (101 ... 116)
NODEID	IN	BYTE	CAN address (1 ... 127)
TRANSFERTYP	IN	BYTE	Type of transfer
INDEX	IN	DWORD	CANopen Index
SUBINDEX	IN	DWORD	CANopen sub index
CANOPENERROR	OUT	DWORD	CANopen error
RETVAL	OUT	WORD	Return value (0 = OK)
BUSY	OUT	BOOL	Function is busy
DATABUFFER	INOUT	ANY	Data Buffer for FC/SFC communication

REQUEST

Control parameter: 1: Initialization of the order

SLOT_MASTER

101...116: slot 1 ... 16 from master at SPEED-Bus

NODELD

Address of the CANopen node (1...127)

TRANSFERTYPE

40h: Read SDO	23h: Write SDO (1 DWORD)
	2Bh: Write SDO (1 WORD)
	2Fh: Write SDO (1 BYTE)

INDEX	CANopen Index
SUBINDEX	CANopen sub index

SLOT_MASTER

0:	System 200 CPU 21xCAN
1...32:	System 200 IM 208CAN
101...115:	System 300S 342-1CA70

CANOPENERROR When no error occurs, *CANOPENERROR* returns 0. In case of an error *CANOPENERROR* contains one of the following error messages that are created by the CAN master:

Code	Description
0503 0000h	Toggle Bit not alternated
0504 0000h	SDO Time out value reached
0504 0001h	Client/server command specify not valid, unknown
0504 0002h	Invalid block size (only block mode)
0504 0003h	Invalid sequence number (only block mode)
0504 0004h	CRC error (only block mode)
0504 0005h	Insufficient memory
0601 0000h	Attempt to read a write only object
0601 0001h	Attempt to write a read only object
0602 0000h	Object does not exist in the object dictionary
0604 0041h	Object cannot be mapped to the PDO
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason
0604 0047h	General internal incompatibility reason in the device
0606 0000h	Access failed because of an hardware error
0607 0010h	Data type does not match, length of service parameter does not match.
0607 0012h	Data type does not match, length of service parameter exceeded.
0607 0013h	Data type does not match, length of service parameter shortfall.
0609 0011h	Sub index does not exist
0609 0030h	Value range of parameter exceeded (only for write access)
0609 0031h	Value of parameter written too high
0609 0032h	Value of parameter written too low
0609 0036h	Maximum value is less than minimum value
0800 0000h	General error
0800 0020h	Data cannot be transferred or stored to the application.

Code	Description
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

RETVAL

When the function has been executed without error, the return value contains the valid length of the response data: 1: BYTE, 2: WORD, 4: DWORD. If an error occurs during execution, the return value contains one of the following error codes.

Code	Description
F021h	Invalid slave address (call parameter equal 0 or higher 127)
F022h	Invalid transfer type (value not equal to 40h, 23h, 2Bh, 2Fh)
F023h	Invalid data length (data buffer too small, at SDO read access this should be at least 4byte, at SDO write access at least 1byte, 2byte or 4byte).
F024h	FC/SFC is not supported.
F025h	Write buffer in CANopen master overflow, service cannot be processed at this time.
F026h	Read buffer in CANopen master overflow, service cannot be processed at this time.
F027h	SDO read or write access with defective response ↳ *'CANOPENERROR' on page 56*.
F028h	SDO timeout (no CANopen station with this node-ID found).

BUSY

As long as *BUSY* = 1, the current order is not finished.

DATABUFFER

- Data area via that the FC/SFC communicates. Set here an ANY pointer of the type Byte.
- SDO read access: Destination area for the read user data.
- SDO write access: Source area for the user data to write.

When the SDO request has been executed without errors, RETVAL contains the length of the valid response data (1, 2 or 4byte) and CANOPENERROR the value 0.

5.4.5 FC/SFC 254 - RW_SBUS - IBS communication

Description

This block serves the INTERBUS-FCs 20x as communication block between INTERBUS master and CPU.

For the usage of the INTERBUS-FCs 20x the FC/SFC 254 must be included in your project as block.

Parameters

Parameter	Declaration	Type	Description
READ/WRITE	IN	Byte	0 = Read, 1 = Write
LADDR	IN	WORD	Logical Address INTERBUS master
IBS_ADDR	IN	WORD	Address INTERBUS master
DATAPOINTER	IN	ANY	Pointer to PLC data
RETVAL	OUT	WORD	Return value (0 = OK)

READ/WRITE

This defines the transfer direction seen from the CPU. *READ* reads the data from the Dual port memory of the INTERBUS master.

LADDR

Enter the address (**L**ogical **Addr**ess) from where on the register of the master is mapped in the CPU. At the start-up of the CPU, the INTERBUS master are stored in the I/O address range of the CPU following the shown formula if no hardware configuration is present:

Start address = 256× (slot-101)+2048

The slot numbers at the SPEED-Bus start with 101 at the left side of the CPU and raises from the right to the left. For example the 1. slot has the address 2048, the 2. the address 2304 etc.

IBS_ADDR

Address in the address range of the INTERBUS master.

DATAPOINTER

Pointer to the data area of the CPU.

RETVAL

Value that the function returns. 0 means OK.

5.5 System Function Blocks

5.5.1 SFB 7 - TIMEMESS - Time measurement

In opposite to the SFC 53, the SFB 7 returns the difference between two calls in µs. With *RESET* = 1 the current timer value is transferred to InstDB. Another call with *RESET* = 0 displays the difference in µs via *VALUE*.

Parameters

Name	Declaration	Type	Comment
RESET	IN	BOOL	*RESET* = 1 start timer
VALUE	OUT	DWORD	Difference in µs

RESET

RESET = 1 transfers the current timer value to InstDB. Here *VALUE* is not influenced.

VALUE

After a call with *RESET* = 0, *VALUE* returns the time difference between the two SFB 7 calls.