15 Status Bits

The status bits are binary "flags" (condition code bits). The CPU uses them for controlling the binary logic operations and sets them during digital processing. You can check these status bits or act upon specific bits. The status bits are combined into a word, the status word. However, you cannot access this status word with LAD or FBD.

15.1 Description of the Status Bits

Table 15.1 shows the available status bits. The CPU uses the binary flags for controlling the binary functions; the digital flags indicate mainly results of arithmetic and mathematical functions.

First check

The /FC status bit steers the binary logic within a logic control system. A bit logic step always starts with /FC = "0" and a binary check instruction, the first check. The first check sets /FC = "1". The first check corresponds in LAD to the first contact in a network, and in FBD to the first binary function input.

Table 15.1 Status Bits

Binary Flags	
/FC	First Check
RLO	Result of logic operation
STA	Status
OR	Status bit OR
BR	Binary result
Digital Flags	
OS	Stored overflow
OV	Overflow
CC0	Condition code (status) bit 0
CC1	Condition code (status) bit 1

A bit logic step ends with a binary value assignment (e.g. of a single coil or an assignment) or with a conditional jump or a block change. These set /FC = "0".

Result of the logic operation (RLO)

The RLO status bit is the intermediate buffer in binary logic operations. In the first check, the CPU transfers the check result to the RLO, combines the check result with the stored RLO on each subsequent check, and stores the result, in turn, in the RLO.

You can store the RLO with the SAVE coil/box in the binary result BR. Memory functions, timers and counters are controlled using the RLO and certain jump functions are executed. The RLO corresponds in LAD to the power flowing in the rung (RLO = "1" is the same as "power flowing").

Status

Status bit STA corresponds to the signal state of the checked binary operand. In the case of memory functions, the value of STA is the same as the written value or (if no write operation takes place, for example if the RLO = "0" or the MCR is active), STA corresponds to the value of the addressed (and unmodified) binary operand.

In the case if edge evaluations FP or FN, the value of the RLO prior to the edge evaluation is stored in STA. All other binary functions set STA = "1".

The STA status bit has no effect on the processing of the LAD or FBD functions.

Status bit OR

Status bit OR stores the result of a fulfilled series circuit or a fulfilled AND condition and indicates to a subsequently processed parallel circuit or OR function that the result has already been determined. All other binary functions reset the OR status bit.

Overflow

Status bit OV indicates a range violation or the use of invalid REAL numbers. The following functions affect the OV status bit: Arithmetic functions, mathematical functions, some conversion functions, REAL comparison functions.

You can check the OV status bit directly.

Stored overflow

The OS status bit stores a set OV status bit: Whenever the CPU sets status bit OV, it also sets status bit OS. However, while the next properly executed operation resets OV, OS remains set. This provides you with the opportunity of evaluating a range violation or an operation with an invalid REAL number, even at a later point in your program.

You can check status bit OS directly. A block change resets the OS status bit.

Status bits CC0 and CC1 (condition code bits)

Status bits CC0 and CC1 provide information on the result of a comparison function, an arithmetic or mathematical function, a word logic operation, or on the bit shifted out by a shift function.

You can check all combinations of CC0 and CC1 directly (see below).

Binary result

LAD and FBD use status bit BR to implement the EN/ENO mechanism for boxes. You can also set, reset or check status bit BR yourself.

15.2 Setting the Status Bits

The digital functions affect status bits CC0, CC1, OV and OS as shown in Table 15.2. You can check these status bits immediately following the function box.

Status bits in INT and DINT calculations

The arithmetic functions with data formats INT and DINT set all digital status bits. A result of zero sets CC0 and CC1 to "0". CC0 = "0" and CC1 = "1" indicates a positive result, CC0 = "1" and CC1 = "0" indicates a negative result. A range violation sets OV and OS (please note the other meaning of CC0 and CC1 in the case of overflow). Division by zero is indicated by all digital status bits being set to "1".

Status bits in REAL calculations

The arithmetic functions with data format REAL and the mathematical functions set all digital status bits. A result of zero sets CC0 and CC1 to "0". CC0 = "0" and CC1 = "1" indicates a positive result, CC0 = "1"and CC1 = "0" indicates a negative result. A range violation sets OV and OS (please note the other meaning of CC0 and CC1 in the case of overflow). An invalid REAL number is indicated when all digital status bits are set to "1".

A REAL number is referred to as "denormalized" if it is represented with reduced accuracy. The exponent is then zero; the absolute value of a denormalized REAL number is less than $1.175\ 494 \times 10^{-38}$. S7-300 CPUs treat denormalized REAL numbers as though they were zero (also see Chapter 3.5.4, "Elementary Data Types").

Status bits for conversion functions

Of the conversion functions, the two's complements affect all digital status bits. In addition, the following conversion functions set status bits OV and OS in the event of an error (range violation or invalid REAL number):

- ▷ I_BCD and DI_BCD:
  Conversion of INT and DINT to BCD
- ▷ CEIL, FLOOR, ROUND, TRUNC:
  Conversion of REAL to DINT

Tabelle 15.2 Setting the Status Bits

INT calculation

The result is:	CC0	CC1	OV	OS
< –32 768 (ADD_I, SUB_I)	0	1	1	1
< –32 768(MUL_I)	1	0	1	1
–32 768 to –1	1	0	0	-
0	0	0	0	-
+1 to +32 767	0	1	0	-
> +32 767 (ADD_I, SUB_I)	1	0	1	1
> +32 767(MUL_I)	0	1	1	1
32 768(DIV_I)	0	1	1	1
(–) 65 536	0	0	1	1
Division by zero	1	1	1	1

DINT calculation

The result is:	CC0	CC1	OV	OS
< –2 147 483 648 (ADD_DI, SUB_DI)	0	1	1	1
< –2 147 483 648 (MUL_DI)	1	0	1	1
–2 147 483 648 to –1	1	0	0	-
0	0	0	0	-
+1 to +2 147 483 647	0	1	0	-
> +2 147 483 647 (ADD_DI, SUB_DI)	1	0	1	1
> +2 147 483 647 (MUL_DI)	0	1	1	1
2 147 483 648(DIV_DI)	0	1	1	1
(–) 4 294 967 296	0	0	1	1
Division by zero (DIV_DI, MOD_DI)	1	1	1	1

REAL calculation

The result is:	CC0	CC1	OV	OS
+ normalized	0	1	0	-
± denormalized	0	0	1	1
± zero	0	0	0	-
– normalized	1	0	0	-
+ infinite (division by zero)	0	1	1	1
– infinite (division by zero)	1	0	1	1
± invalid REAL number	1	1	1	1

Comparison

The result is:	CC0	CC1	OV	OS
equal to	0	0	0	-
greater than	0	1	0	-
less than	1	0	0	-
invalid REAL number	1	1	1	1

Conversion NEG_I

The result is:	CC0	CC1	OV	OS
+1 to +32 767	0	1	0	-
0	0	0	0	-
–1 to –32 767	1	0	0	-
(–) 32 768	1	0	1	1

Conversion NEG_D

The result is:	CC0	CC1	OV	OS
+1 to +2 147 483 647	0	1	0	-
0	0	0	0	-
–1 to –2 147 483 647	1	0	0	-
(–) 2 147 483 648	1	0	1	1

Shift function

The shifted out bit is:	CC0	CC1	OV	OS
"0"	0	0	0	-
"1"	0	1	0	-
with shift number 0	-	-	-	-

Word logic

The result is:	CC0	CC1	OV	OS
zero	0	0	0	-
not zero	0	1	0	-

Status bits for comparison functions

The comparison functions set the CC0 and CC1 status bits. The flags are set independently of the executed comparison function.

Status bits for shift functions

In the case of the shift functions, the signal state of the last bit to be shifted out is transferred to status bit CC1. CC0 and OV are reset.

Status bits for word logic

If the result of the word logic operation is zero (all bits are "0"), CC1 is reset; if at least one bit in the result is "1", CC1 is set. CC0 and OV are reset.

15.3 Evaluating the Status Bits

LAD: You can use the normally-open (NO) and the normally-closed (NC) contact to check the digital status bits and the binary result. Figure 15.1 shows the check with a normally-open contact. The check with a normally-closed contact returns the negated check result. You can handle the NO and NC contacts for evaluating the status bits in exactly the same way as the "normal" contacts. You can find examples of evaluating the status bits (FB 115 in the "Program Flow Control" program in the "LAD Book" library that you can download from the publisher's Website (see page 8).

LAD representation	FBD representation	A current flows, or the check is fulfilled when
>0	>0	Result greater than zero [(CC0=0) & (CC1=1)]
>=0	>=0	Result greater than or equal to zero [(CC0=0)]
<0	<0	Result less than zero [(CC0=1) & (CC1=0)]
<=0	<=0	Result less than or equal to zero [(CC1=0)]
<>0	<>0	Result not equal to zero [(CC0=0) & (CC1=1) v (CC0=1) & (CC1=0)]
==0	==0	Result equal to zero [(CC0=0) & (CC1=0)]
UO	UO	Result invalid (unordered) [(CC0=1) & (CC1=1)]
OV	OV	Overflow [OV=1]
OS	OS	Stored overflow [OS=1]
BR	BR	Binary result [BR=1]

Figure 15.1 Evaluating the Status Bits

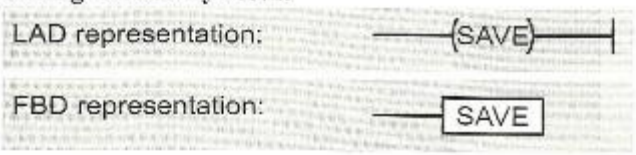
FBD: Direct or negated checking of the digital status bits and the binary result is possible. This is shown in Figure 15.1 by direct checking for signal state "1". The check for signal state "0" returns the negated check result. The checks for evaluation of the status bits can be treated exactly the same as the "normal" checks for binary operands. You can find examples of evaluating the status bits (FB 115 in the program "Program Flow Control") in the "FBD_ Book" library that you can download from the publisher's Website (see page 8) .

In the case of incremental programming, you will find these checks in the Program Elements Catalog (with VIEW → OVERVIEWS [Ctrl - K] or INSERT → PROGRAM ELEMENTS) under "Status Bits".

15.4 Using the Binary Result

15.4.1 Setting the Binary Result BR

Saving the binary result



SAVE coil in a rung (LAD)

You store the RLO in the binary result using the SAVE coil. If power flows into the SAVE coil, BR is set, otherwise it is reset. You program the SAVE coil in the same way as a "single" coil without binary operands.

Please note that the SAVE coil does not terminate the logic operation (the /FC status bit is not set to "0"). This means that the logic operation preceding the SAVE coil is also the preceding logic operation for the next network.

The SAVE coil cannot be programmed in conjunction with a T-branch.

SAVE box in a logic circuit (FBD)

With the SAVE box, you save the RLO in the binary result. If the RLO is "1" before the SAVE box, BR is set; otherwise, BR is reset. You program the SAVE box the same way you program an assign box without binary operands.

Please note that the SAVE box does not terminate a logic operation (status bit /FC is not set to "0"). The next network thus begins with an "open" logic operation.

The SAVE box is not permitted after a T-branch.

Controlling the binary result

LAD and FBD affect even the binary result in order to control the ENO output (Figure 15.2). If enable output ENO is wired, its signal state is the same as that of the BR. In certain cases ("BR corresponds to function"), the LAD or FBD function executed sets the binary result as follows:

- BR := "1"
  for MOVE, for the shift functions and for the word logic operations
- BR := OV
  for the arithmetic and mathematical functions
- BR := OV or "1"
  for the conversion functions
- BR := BR of the called block in the case of block calls

15.4.2 Main Rung, EN/ENO Mechanism

In programming languages LAD and FBD, many boxes have an enable input EN and an enable output ENO. If the enable input is "1", the function in the box is processed. When the box is processed correctly, the enable output also has signal state "1". If an error occurs during processing of a box (for example, overflow during execution of an arithmetic function), ENO is set to "0". If EN has signal state "0", ENO is also set to "0".

These characteristics of EN and ENO can be used in order to connect several boxes together in a chain, with the enable output leading to the enable input of the next box (Figure 15.3). This means, for example, that the entire chain can be "disabled" (no boxes are processed if *input I 1.0* in the example has signal state "0") or the rest of the chain is no longer processed if one box signals an error.

Is ENO switched ?					
YES			NO		
Is EN switched ?			Is EN switched ?		
YES		NO	YES		NO
Is EN == "1" ?			Is EN == "1" ?		
YES	NO		YES	NO	
BR corre-sponds to func-tion	BR := "0"	BR corre-sponds to func-tion	BR := "1"	BR := "0"	BR not affected

Figure 15.2 General Schematic for Setting the Binary Result

The input EN and the output ENO are not block parameters but statement sequences that the Program Editor itself generates prior to and following all boxes (also in the case of functions and function blocks). The Program Editor uses the binary result to store the signal state at EN while the block is being processed or to check the error flag from the box.

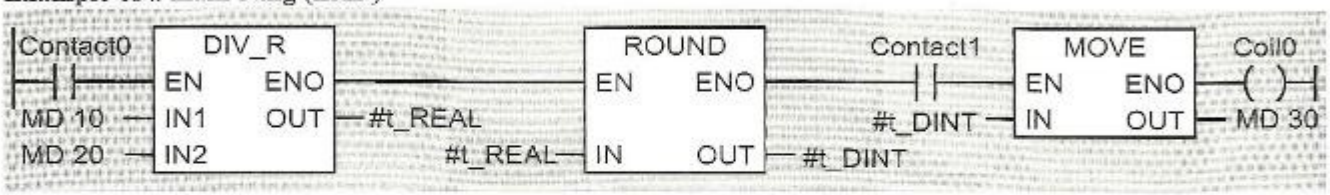
15.4.3 ENO in the Case of User-written Blocks

The Program Editor provides the call of your own blocks with the enable input EN and the enable output ENO. You can use the enable input EN to call the block conditionally. You can use the ENO output, for example, to signal a group error (signal state "1", if the block has been properly processed; "0" if an error has occurred during processing of the block). All system blocks also signal group errors via BR.

You control the ENO output with the binary result BR. The ENO output has the same signal state that BR has when the block is exited.

For example, BR could be set to "1" at the start of a block. If an error then occurs during processing of the block, for example if a result exceeds the fixed range so that further processing must be prevented, set the binary result to "0" with the SAVE coil/box and jump to the end of the block where the block will be exited (in the event of an error, the condition must supply signal state "0"). Please note that the RET coil/box sets the BR to "1" if you exit the block via this coil/box.

Example of a main rung (LAD)



Example of a series connection of boxes (FBD)

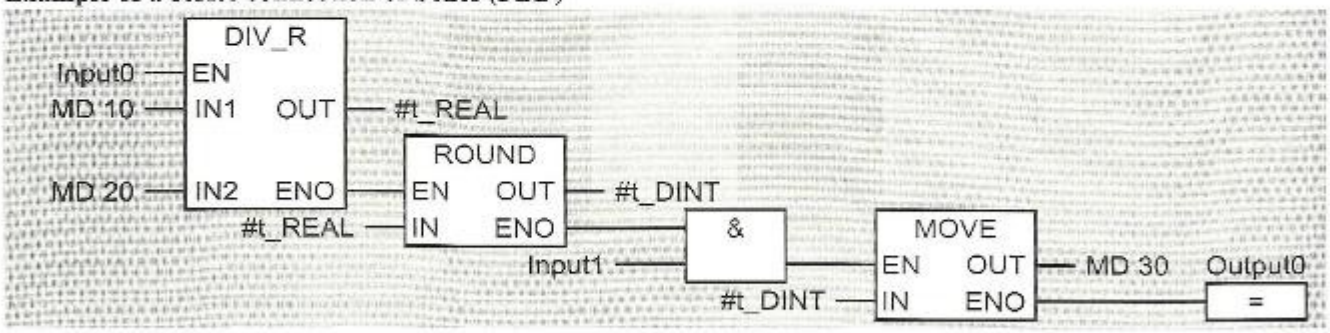


Figure 15.3 Example for the Series Connection of EN and ENO

16 Jump Functions

You can use jump functions to interrupt the linear flow of the program and continue at another point in the block. This program branching can be executed unconditionally or dependent on the RLO.

16.1 Processing a Jump Function

Representation

LAD representation

Jump if RLO = "1"	Dest (JMP)
Jump if RLO = "0"	Destn (JMPN)
Entry, jump label	Dest

FBD representation

Jump if RLO = "1"	Dest JMP
Jump if RLO = "0"	Dest JMPN
Entry, jump label	Dest

A jump function consists of the jump operation in the form of a coil (LAD) or box (FBD) and a jump label designating the program location at which processing is to continue after the jump. The jump label is above the jump operation.

Jumps cannot be programmed in conjunction with a T-branch.

A jump label consists of up to 4 characters that can include letters, digits, and the underscore. It begins with letter. A jump label in a box designates the network that is to be processed following completion of the jump operation. The box with the jump label must be at the start of a network ("LABEL" in the Program Elements Catalog).

In the case of incremental programming, you will find the jump functions in the Program Elements Catalog (with VIEW → OVERVIEWS [Ctrl - K] or INSERT → PROGRAM ELEMENTS) under "Jumps".

Function

A jump is either always executed (absolute or unconditional jump) or it is executed depending on the result of the logic operation (RLO) (conditional jump). In the case of a jump dependent on the RLO, you can decide whether the jump is to be executed if the RLO is "1" or if the RLO is "0".

You can execute both forward (in the direction of program processing; in the direction of ascending network numbers) and backward jumps. The jump can take place only within a block, that is, the jump destination must be in the same block as the jump function. If you use the Master Control Relay (MCR), the jump label must be in the same MCR zone or in the same MCR area as the jump function.

The jump destination must be designated unambiguously, that is, you must only assign a jump label once in a block. The jump destination can be jumped to from several locations.

The Program Editor stores the names of the jump labels in the non-executable section of the relevant blocks on the programming device's data medium. Only the widths of the jumps (jump displacement) are stored in the CPU's work memory (in the compiled block). When program modifications are made online to blocks in the CPU, these modifications must therefore always be updated on the program-

ming device's data medium in order to retain the original label names. If this update is not made, or if blocks are transferred from the CPU to the programming device, the non-executable block sections will be overwritten or deleted. On the display or the printout, the Editor then generates replacement symbols for the jump labels (M001, M002 etc.).

16.2 Unconditional Jump

The unconditional jump, which is the jump that is always executed, is jump function JMP, whose coil is connected to the left power rail (LAD) or whose box has no preceding logic operation (FBD). This jump function is always executed when encountered in the program. The CPU interrupts the linear flow of the program and continues in the network designated by the jump label.

Example (Figure 16.1 and Figure 16.2): In Network 3, there is an unconditional jump to jump label M2. After this network has been processed, the CPU continues program execution at jump label M2 in Network 5. A jump from another program location is required in order to process Network 4.

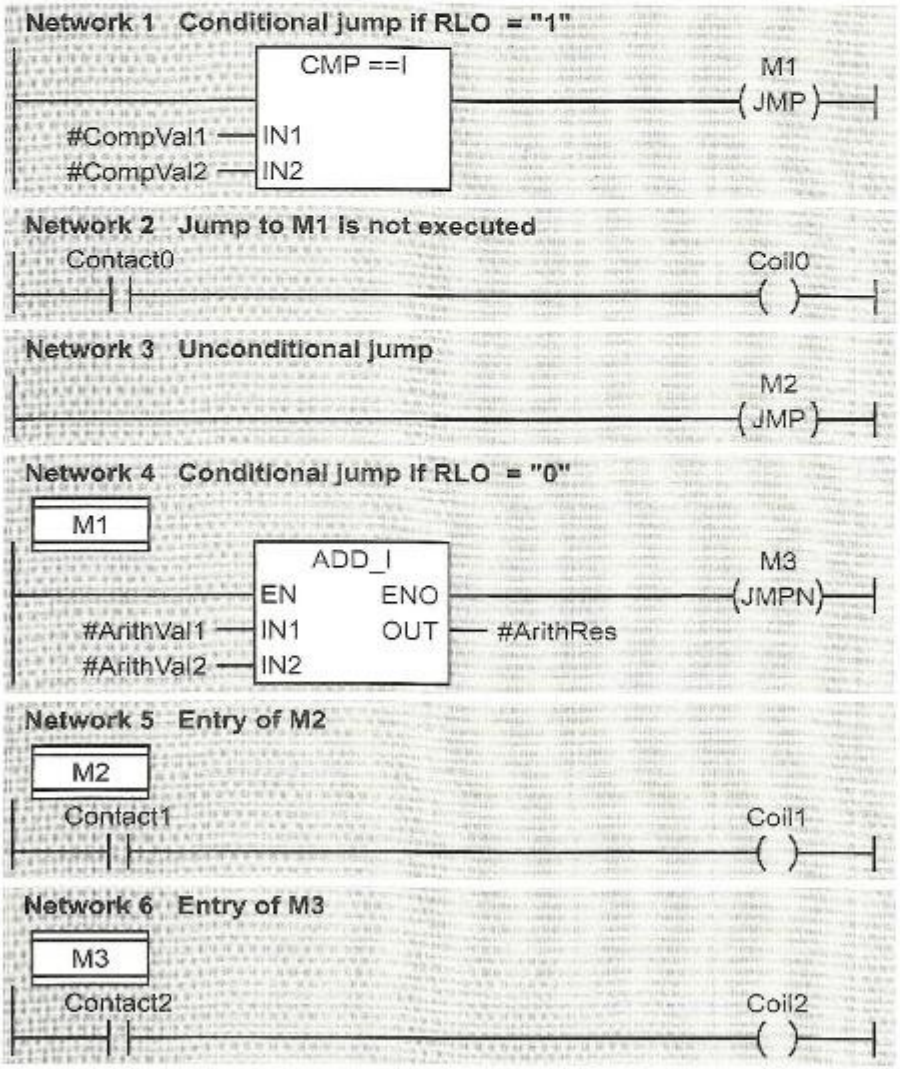


Figure 16.1 Jump Functions Example (LAD)

16.3 Jump if RLO = "1"

The conditional jump if RLO = "1" is jump function JMP, whose coil is not directly connected to the left power rail (LAD) or whose box has a preceding logic operation (FBD). The logic operation preceding this coil can be implemented in any way. If the RLO = "1" (if the preceding logic operation is fulfilled), the CPU interrupts the linear flow of the program and continues in the network designated by the jump label. If the preceding logic operation is not fulfilled, the CPU continues program execution in the next network.

Example (Figure 16.1 and Figure 16.2): If the compare condition in Network 1 is fulfilled, program execution continues in Network 4. If the compare condition is not fulfilled, the next network, which is Network 2, is processed.

16.4 Jump if RLO = "0"

The conditional jump if RLO = "0" is jump function JMPN, whose coil is not directly connected to the left power rail (LAD) or whose box has a preceding logic operation (FBD). The logic operation preceding this coil can be implemented in any way. If the RLO = "0" (if the preceding logic operation is not fulfilled), the CPU interrupts the linear flow of the program and continues in the network designated by the jump label. If the preceding logic operation is fulfilled, the CPU continues program execution in the next network.

Example (Figure 16.1 and Figure 16.2): If the adder in Network 4 signals an error, the jump to Network 6 (jump label M3) is executed. If no error is signaled, Network 5 is the next to be processed.

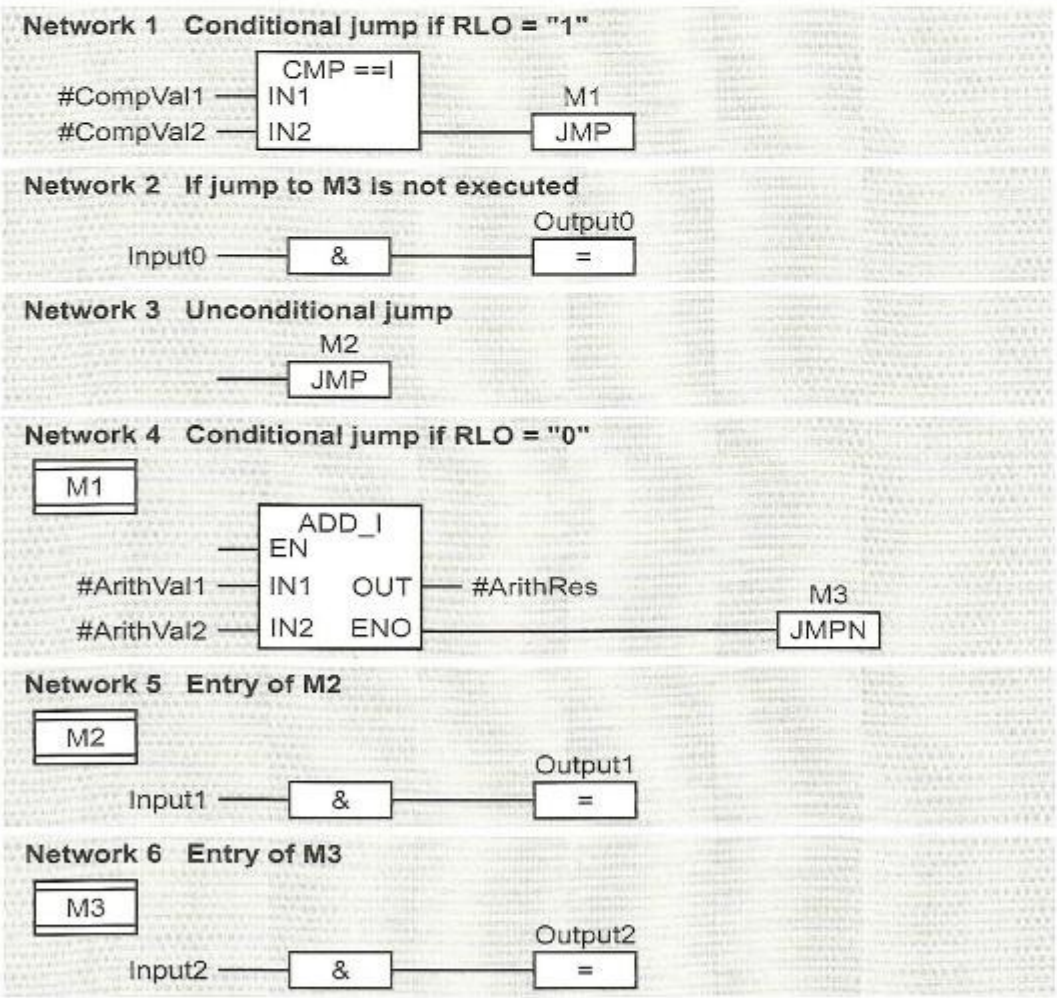


Figure 16.2 Jump Functions Example (FBD)

17 Master Control Relay

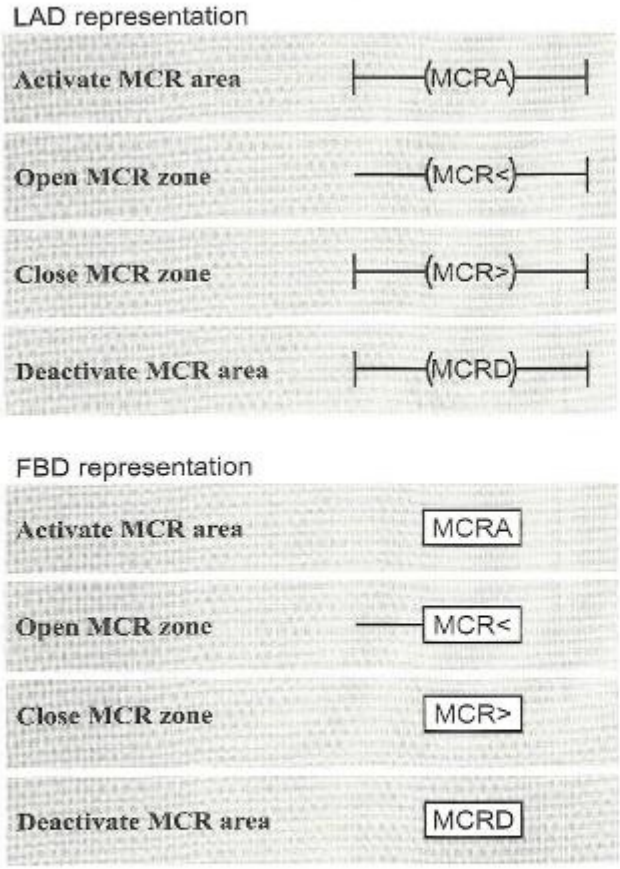
In contact control systems, a Master Control Relay activates or deactivates a section of the control system that can consist of one or more rungs.

A deactivated rung

- ▷ deenergizes all non-retentive contactors and
- ▷ retains the states of retentive contactors.

You cannot change the state of the contactors again until the Master Control Relay (MCR) is active.

Please note that deenergizing with the "software" Master Control Relay is no substitute for an EMERGENCY OFF or safety facility! Treat Master Control Relay switching in exactly the same way as switching with a memory function!



With MCRA and MCRD, you designate an area in your program in which MCR dependency is to take effect. Within this area, you use MCR< and MCR> to define one or more zones in which the MCR dependency can be enabled and disabled. You can also nest the MCR zones. The result of the logic operation (RLO) immediately preceding an MCR zone enables or disables MCR dependency within that zone.

17.1 MCR Dependency

MCR dependency affects coils and boxes. If MCR dependency is enabled (corresponds to Master Control Relay disabled)

- ▷ a single coil or assign box and a midline output set the binary operand to signal state "0" (following the midline output, the RLO is then = "0", that is, power no longer flows)
- ▷ a set and reset coil or box no longer affect the signal state of the binary operand ("freeze it")
- ▷ an SR and RS box no longer affect the signal state of the binary operand ("freeze it")
- ▷ a transfer operation writes zero to the digital operand (every function output of a box of digital data type then writes zero to the operand or to the variable)

Additional to the direct influencing of an operand the RLO is then "0" (power no longer flows) behind a midline output and a T-branch.

Some LAD and FBD functions use transfer statements (invisible to the user). Since a transfer statement writes the value zero if MCR dependency is switched on, the corresponding function can no longer be guaranteed

You must exclude the following program sections from MCR dependency otherwise the CPU will go to STOP or undefined runtime behavior can occur:

- Block calls with block parameters
- Accesses to block parameters that are parameter types (e.g. BLOCK_DB)
- Accesses to block parameters that are components or elements of complex data types or UDTs

You enable MCR dependency in a zone if the RLO is "0" immediately prior to opening the zone (analogous to disabling the Master Control Relay). If you open an MCR zone with RLO "1" (Master Control Relay enabled), processing within this MCR zone takes place without MCR dependency. MCR dependency is effective only within an MCR zone.

In the case of incremental programming, you will find the MCR functions in the Program Elements Catalog (with VIEW → OVERVIEWS [Ctrl - K] or INSERT → PROGRAM ELEMENTS) under "Program Control".

17.2 MCR Area

To be able to use the characteristic features of the Master Control Relay, define an MCR area with MCRA (start) and MCRD (end of the MCR area). MCR dependency is active within an MCR area, but not yet enabled (Figure 17.1).

The MCRA coil/box and the MCRD coil/box always stand alone in separate networks.

If you call a block within an MCR area, MCR dependency is deactivated in the block called. An MCR area only starts again with the MCRA coil/box. When a block is exited, MCR dependency is set as it was before the block was called, regardless of the MCR dependency with which the called block was exited.

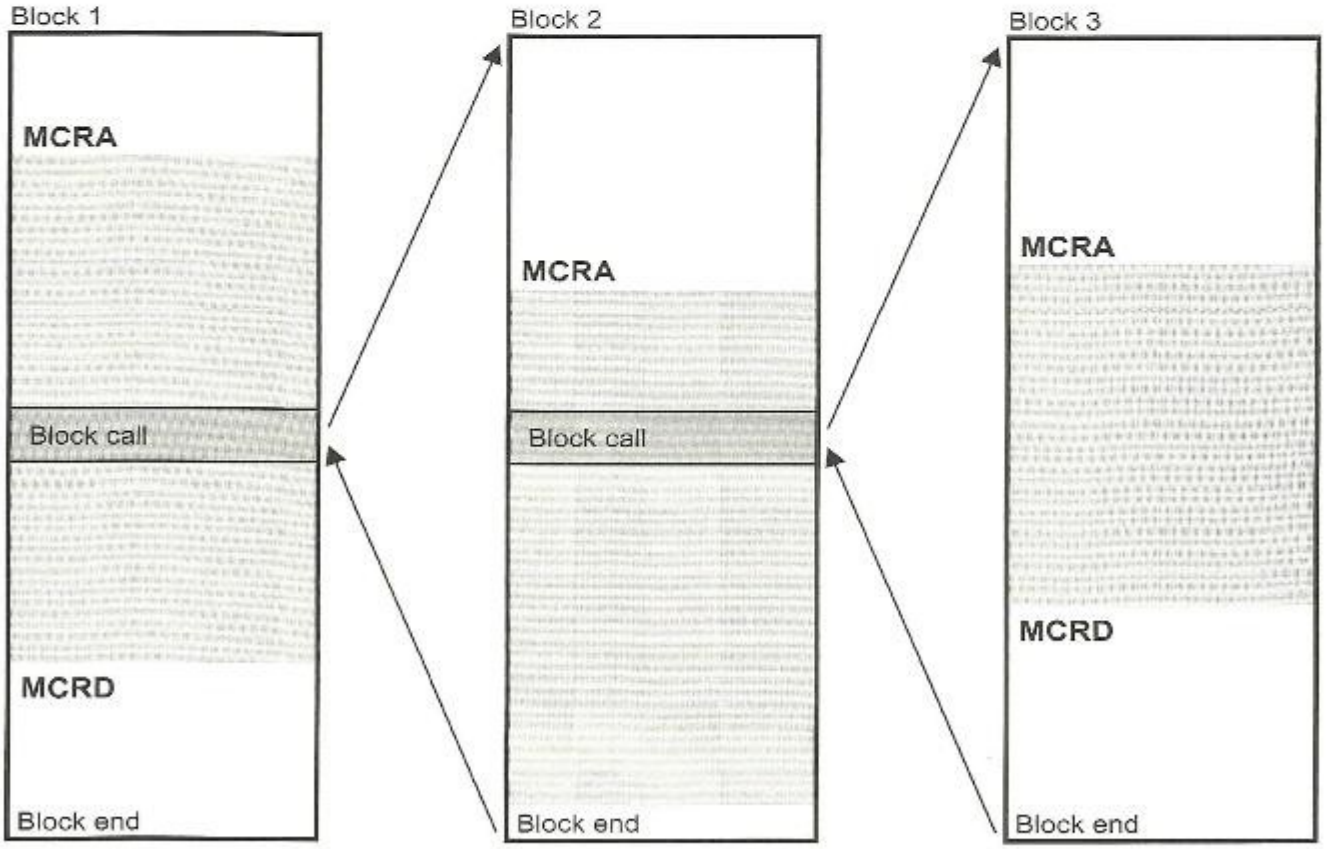


Figure 17.1 MCR Area in the Case of Block Change

17.3 MCR Zone

LAD: You define an MCR zone with the MCR< coil (start) and the MCR> coil (end of the MCR zone). The MCR< coil requires a preceding logic operation; the MCR> coil is connected directly to the left power rail. Both coils terminate a rung. Within this zone, you control MCR dependency with the RLO prior to the MCR< coil: If power flows into the coil, MCR dependency is disabled ("normal" processing); if power does not flow into the coil, MCR dependency is enabled.

FBD: You define an MCR zone with the MCR< box at the beginning and the MCR> box at the end of the MCR zone. The MCR< box requires a preceding logic operation; the MCR> box stands alone in a network. Within this zone, you control MCR dependency with the RLO preceding the MCR< box: If it is "1", MCR dependency is disabled ("normal" processing); if it is "0", MCR dependency is enabled.

You can open an MCR zone within another MCR zone. The nesting depth for MCR zones is 8; that is, you can open a zone up to eight times before you close a zone (Figure 17.2).

You control the MCR dependency of a nested MCR zone with the RLO on opening the zone. However, if MCR dependency is enabled in a "supraordinate" zone, you cannot disable MCR dependency in a "subordinate" MCR zone. The Master Control Relay of the first MCR zone controls the MCR dependency in all nested.

A block call within an MCR zone does not change the nesting depth of an MCR zone. The program in the called block is still in the MCR zone that was open when the block was called (and is controlled form here). However, you must reactivate MCR dependency in a called block by opening the MCR area.

In Figure 17.3, memory bits M 10.0 and M 11.0 control the MCR dependencies. With memory bit M 10.0, you can enable MCR dependency in both zones (with "0") regard-

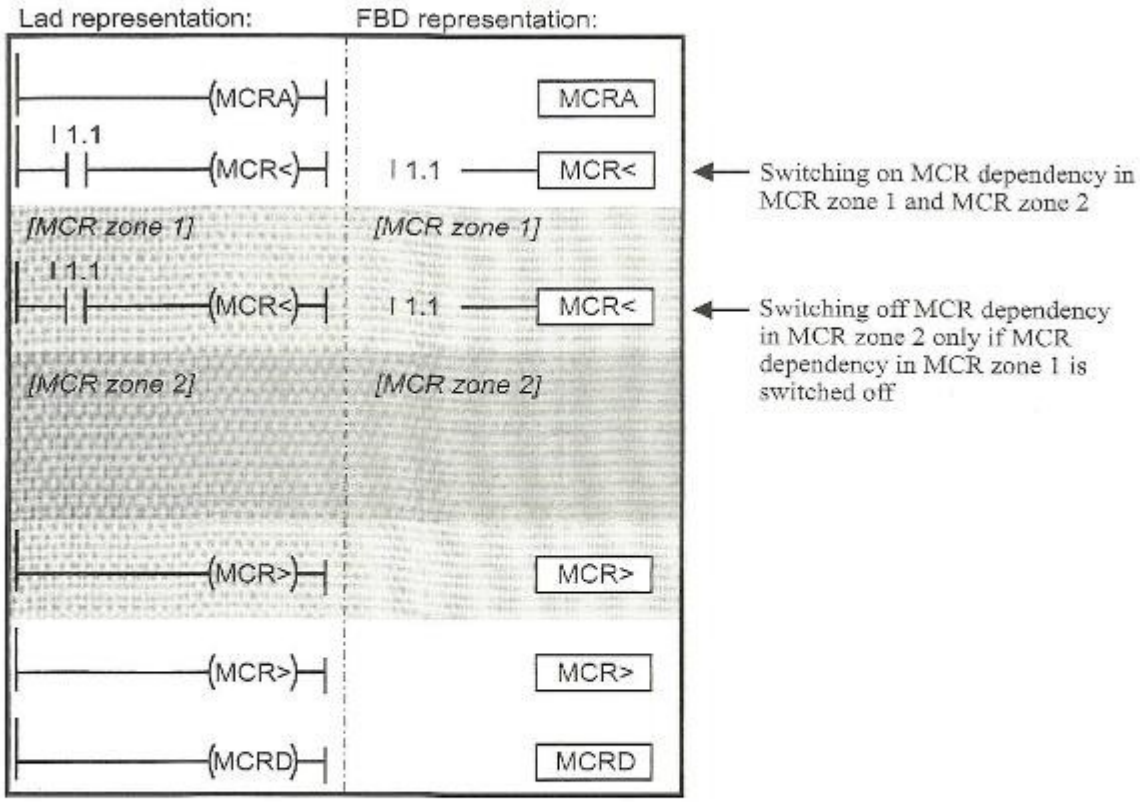


Figure 17.2 MCR Dependency in Nested MCR Zones

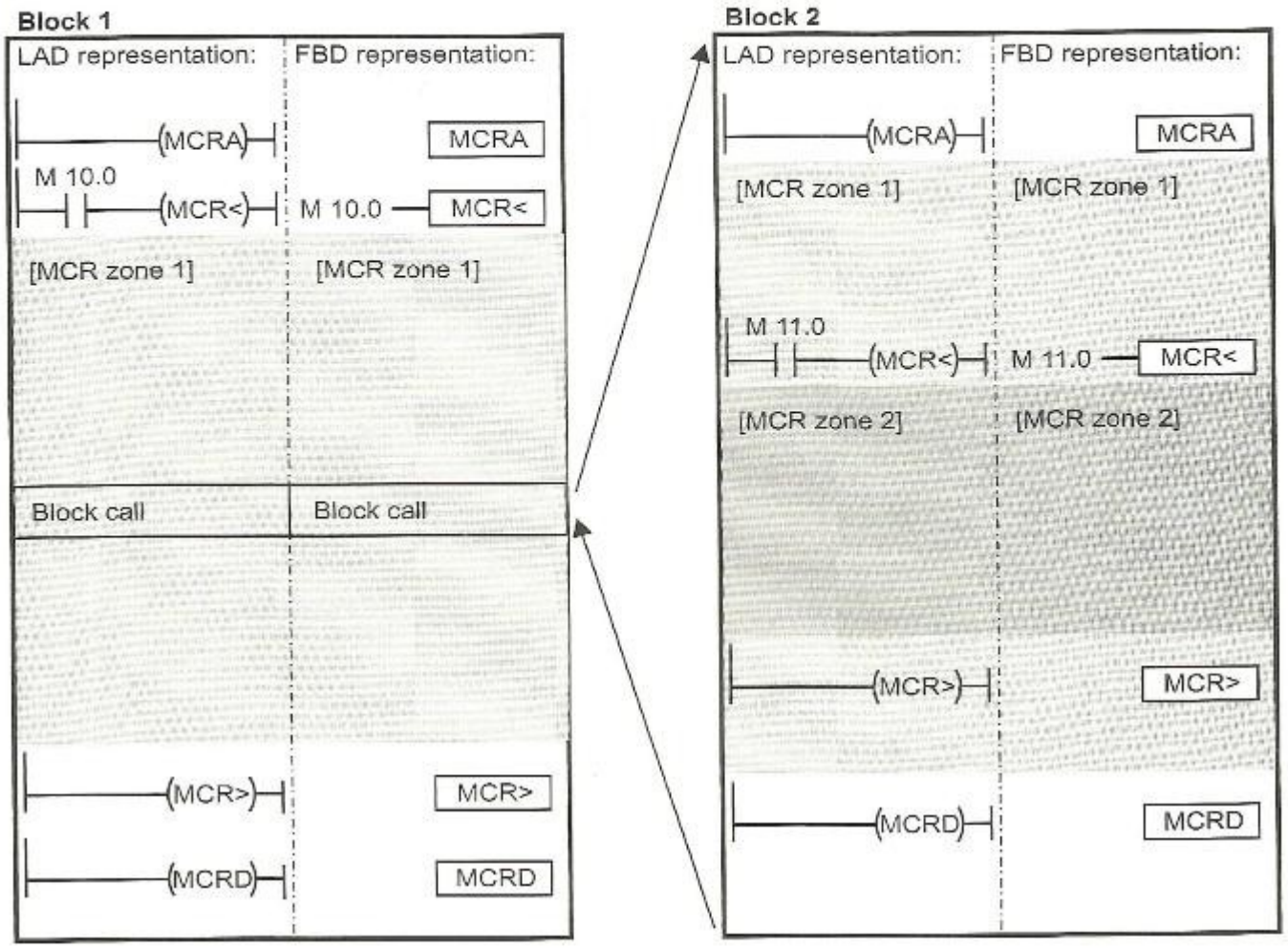


Figure 17.3 MCR Zones in the Case of Block Change

less of the signal state of memory bit M 11.0. If the MCR dependency for zone 1 is disabled with M 10.0 = "1", you can control the MCR dependency of zone 2 with memory bit M 11.0 (Table 17.1).

17.4 Setting and Resetting I/O Bits

Despite enabled MCR dependency, you can set or reset the bits of an I/O area with the system functions. A requirement for this is that the bits to be controlled are in the process-image output table or a process-image output table has been defined for the I/O area to be controlled.

The system function **SFC 79 SET** is available for setting and **SFC 80 RSET** for resetting the I/O bits (Table 17.2). You call these system functions in an MCR zone. The system functions are only effective if MCR dependency is enabled; if MCR dependency is disabled, calling these SFCs has no effect.

Table 17.1 MCR Dependency in the Case of Nested MCR Zones (Example)

M 10.0	M 11.0	Zone 1	Zone 2
"1"	"1"	No MCR dependency	
"1"	"0"	No MCR dependency	MCR dependency enabled
"0"	"1" or "0"	MCR dependency enabled	

Table 17.2 Parameters of the SFCs for Controlling the I/O Bits

SFC	Parameter	Declaration	Data Type	Assignment, Description
79	N	INPUT	INT	Number of bits to be set
	RET_VAL	OUTPUT	INT	Error information
	SA	OUTPUT	POINTER	Pointer to the first bit to be set
80	N	INPUT	INT	Number of bits to be reset
	RET_VAL	OUTPUT	INT	Error information
	SA	OUTPUT	POINTER	Pointer to the first bit to be reset

Setting and resetting the I/O bits also simultaneously updates the process-image output table. The I/O are affected byte by byte. The bits not selected with the SFCs (in the first and in the last byte) retain the signal states as they are currently available in the process image.

You can find examples of the Master Control Relay and of the system functions SFC 79 and SFC 80 in function block FB 117 in the libraries "LAD_Book" and "FBD_Book" in the program "Program Flow Control" that you can download from the publisher's Website (see page 8).

18 Block Functions

In this chapter, you will learn how to call and terminate code blocks and how to work with operands from data blocks. The next chapter then deals with using block parameters.

18.1 Block Functions for Code Blocks

Block functions for code blocks include instructions for calling and terminating blocks (Figure 18.1). Code blocks are called with the call box. If functions or system functions have no block parameters, they can also be called with the CALL coil or with the CALL box. In both cases, a preceding logic operation enabling a conditional call (call dependent on conditions) is permissible. The block end function RET always requires a preceding logic operation.

In addition to the block change, the call box also contains the transfer of block parameters. When function blocks are called, it also opens the instance data block. The CALL coil/box is no more than a change to another block and is only meaningful (and permissible) in the case of functions and system functions.

After a block has been terminated, and following the call function, the CPU continues program execution in the block that made the call (the calling block). If an organization block is terminated, the CPU returns to the operating system.

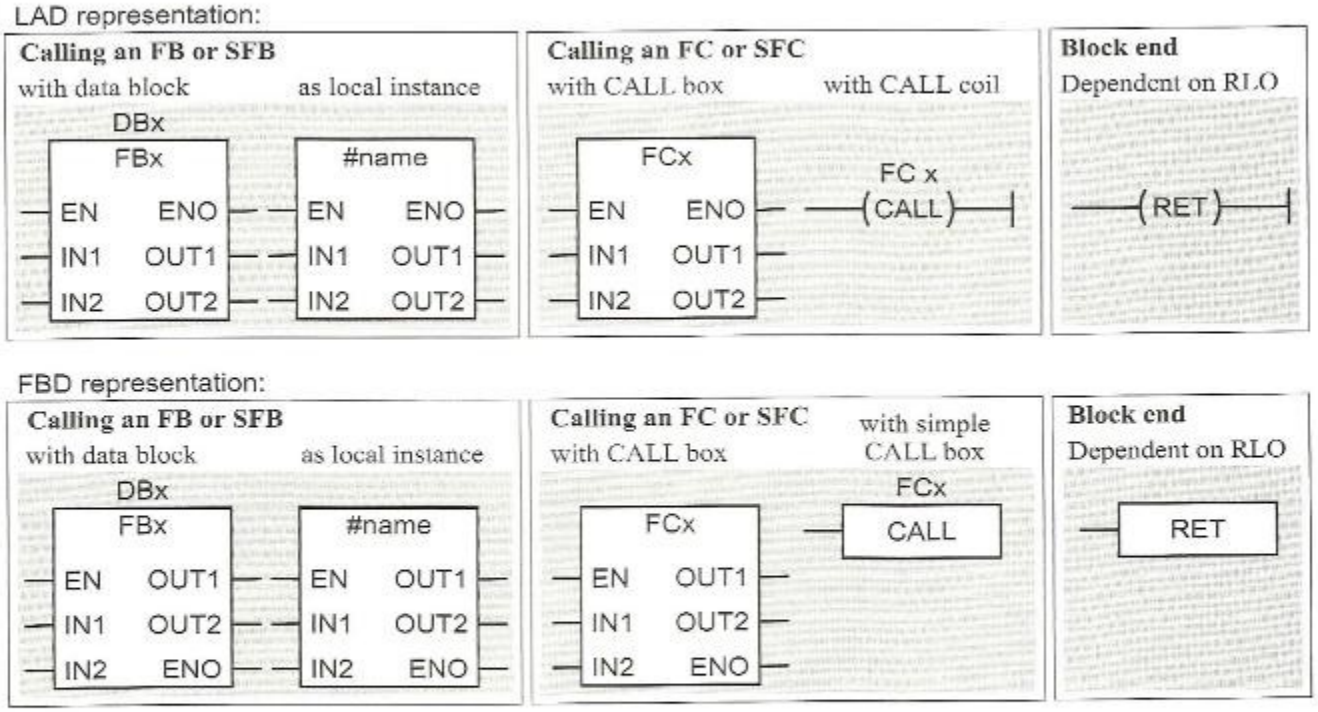


Figure 18.1 Block Functions for Code Blocks

In the case of incremental programming, you will find the CALL coil/box and the RET coil/box in the Program Elements Catalog (with VIEW → OVERVIEWS [Ctrl - K] or INSERT → PROGRAM ELEMENTS) under "Program Control"; you insert block calls with call boxes into your program when you select blocks from "FC/FB/SFC/SFB blocks", "Multiple Instances" or "Libraries".

18.1.1 Block Calls: General

If a code block is to be processed, it must be "called". Figure 18.2 gives an example for calling function FC 10 in organization block OB 1.

A block call consists of the call box that contains the address of the called block (here: FC 10), the enable input EN, the enable output ENO, and any block parameters. Following processing of the call function, the CPU continues program execution in the called block. The block is processed to the end or until a block end function is encountered. The CPU then returns to the calling block (here: OB 1) and continues processing this block after the call box.

The information the CPU requires to find its way back to the calling block is stored in the block stack (B stack). With every new block call, a new stack element is created that

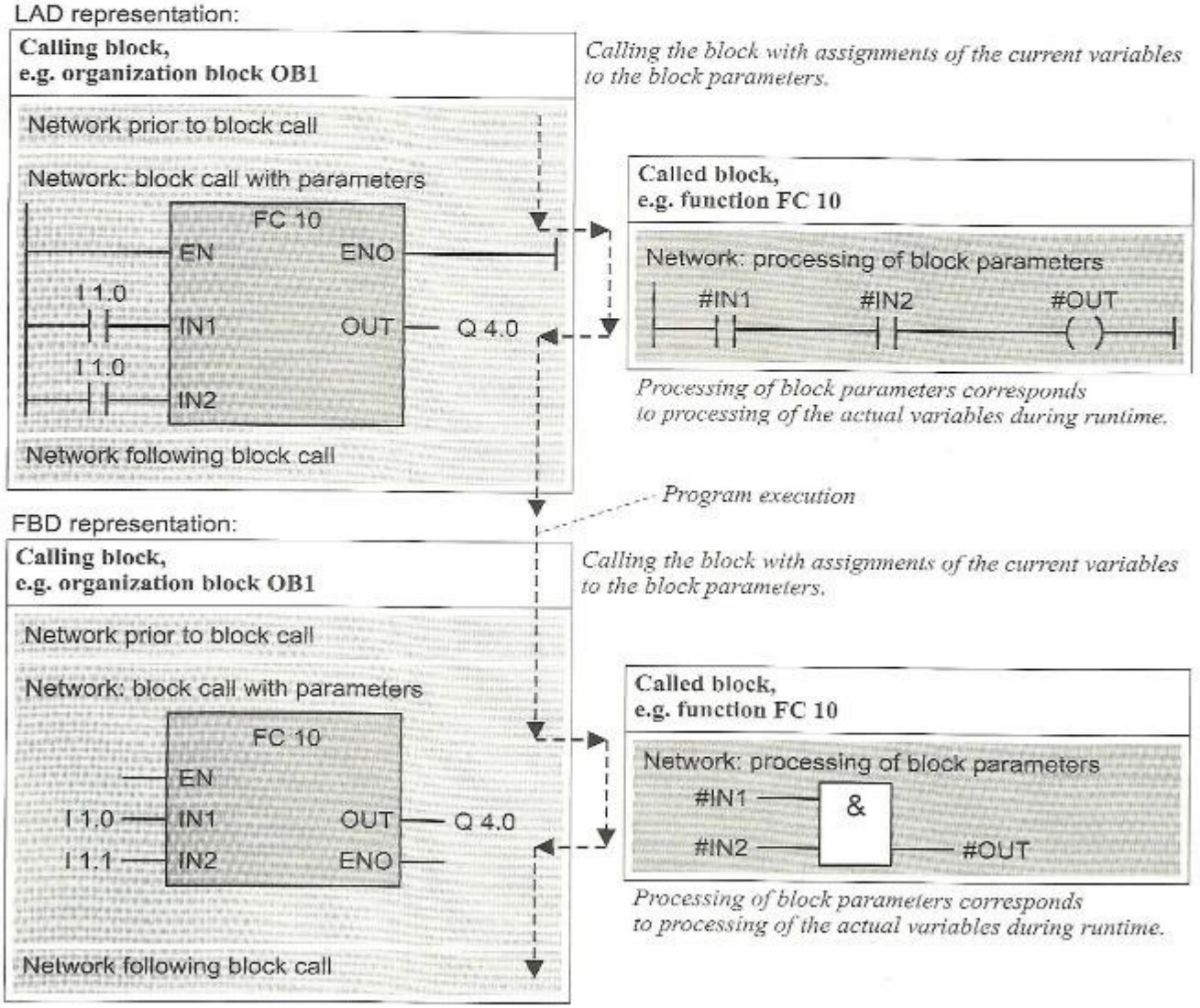


Figure 18.2 Example of a Block Call

includes the return address, the contents of the data block register and the address of the local data stack of the calling block. If the CPU goes to the Stop state as a result of an error, you can use the programming device to see from the contents of the B stack which blocks were processed up to the error.

You can transfer data to and from the called block for processing. These data are transferred via block parameters. With the call box, you can also call blocks without block parameters.

18.1.2 Call Box

You use the call box to call FBs, FCs, SFBs and SFCs. (You cannot call organization blocks since they are event-driven and are started by the operating system.)

You can use the EN input to make the block call subject to conditions. If the EN input is connected direct to the left power rail, the call is absolute; it is always executed. If there is a logic operation preceding EN, the block call is only executed if the preceding logic operation is fulfilled. The ENO output has the same signal state as the binary result BR on exiting the called block.

<table>
<tr><td colspan="3">IF EN == "1" or not connected</td></tr>
<tr><td colspan="2">THEN</td><td>ELSE</td></tr>
<tr><td colspan="2">Called block is processed</td><td rowspan="3">Called block is not processed</td></tr>
<tr><td colspan="2">IF called block returns BR = "1"</td></tr>
<tr><td>THEN</td><td>ELSE</td></tr>
<tr><td>ENO := "1"</td><td>ENO := "0"</td><td>ENO := "0"</td></tr>
</table>

You label the parameters of the called block with the absolute or symbolic operand current for the call. If a parameter is of data type BOOL, precede this parameter with

- a contact or a rung (LAD) or
- a binary variable or a binary logic operation (FBD).

A Boolean output parameter cannot be combined further.

LAD: You can arrange several call boxes in series, connecting them with each other via EN or ENO. You can only insert a call box in a parallel rung if it is connected directly to the left power rail.

FBD: You can connect call boxes in series by connecting the ENO output of one box with the EN input of the next. The ENO outputs of several boxes can be combined with AND or OR.

MCR dependency is de-activated when a block is called. The MCR is disabled in the called block regardless of whether the MCR was enabled or disabled prior to the block call. When exiting a block, MCR dependency assumes the same setting that it had prior to the block call.

Depending on the block parameters, you can modify the contents of the data block registers when the block change is made. If the *called* block is a function block, the instance block is always opened in this block via the DI register. If the *calling* block is a function block, the contents of the DI register (the instance data block) are retained after the block call. The contents of the DB register depend, among other things, on the block parameters that were passed.

Calling function blocks

You call a function block by selecting the relevant function block from the Program Elements Catalog under "FB Blocks". Prerequisite is that the function block to be called must already be in the user program. You write the instance data block belonging to the call above the box. Both blocks (function block and instance data block) can have absolute or symbolic addresses.

In the case of function blocks, you do not need to initialize all block parameters at the call. The uninitialized block parameters retain their current value. However, block parameters saved as pointers should at least be initialized when called for the first time so that meaningful values are entered here (see Chapter 19.3, "Actual Parameters").

You can also call "function blocks with multiple instance capability" within other "function blocks with multiple instance capability" as local instance. In doing so, the called function block uses the instance data block of the calling function block as the store for its local data. Prior to the call, you declare the local instance in the static local data of the calling function

block (the block you are currently programming). The local instance is called by selecting one of the available local instances under 'Multiple Instances' in the Program Elements Catalog; it is not necessary to specify an instance data block (see also Chapter 18.1.6, "Static Local Data").

Calling functions

You call a function by selecting the relevant function under "FC Blocks" in the Program Elements Catalog. The function can have an absolute or a symbolic address.

When you call functions, you must initialize all available parameters.

Calling functions with a function value takes exactly the same form as calling functions with no function value. Only the first output parameter, corresponding to the function value, has the name RET_VAL.

Calling system blocks

The CPU operating system contains system functions (SFCs) and system function blocks (SFBs) that you can use. The number and type of system blocks depends on the CPU. You can call all system blocks with the call box.

You call a system function block in the same way as one you have written yourself; set up the associated instance data block in the work memory with the same data type as the SFB.

You call a system function in the same way as a function you have written yourself.

System blocks exist only in the CPU operating system. If you want to call system blocks during offline programming, the Program Editor needs a description of the call interface so it can initialize the parameters. You will find this interface description under *System Function Blocks* on the library named *Standard Library*. From here, the Program Editor copies the interface description to the offline block container when you call a system block. The interface description thus copied then appears as "normal" block object.

The Program Elements Catalog provides the system blocks currently available in the user program under "SFC Blocks" or "SFB Blocks".

You can, for example, select a system block from the Program Elements Catalog with the mouse and drag it to the block currently being processed block, where it is then called. At the same time, this block (or, more precisely: its interface description) is copied into the block container.

Block calls in sequence (FBD)

If you connect block boxes in sequence in FBD, and have to "pass on" binary signals from one box to the next, you must observe the processing sequence of the binary logic operations by the program editor. The binary operands and logic operations are initially processed prior to the box inputs, starting with the "last" block box; the intermediate binary results are saved in the temporary local data (not shown). Following this, the program editor processes the boxes themselves and their binary outputs, commencing with the "first" box.

Figure 18.3 clarifies the processing sequence. The OR box with the three inputs is initially processed prior to the right block, then the OR box with the two inputs, followed by the left block call (control of Pump1) and finally the right block call (Pump2). The temporary local variable *#t_P1_Fault* would therefore be scanned first (on the OR box with undefined signal state) and then set (at the binary output of the first block call). This could result in falsification of the result of the logic operation.

Remedy: use a global variable (e.g. a bit memory or a data bit; in this case, a change in status at the output of the first block would only be considered in the next program cycle at the input of the second block) for passing on the signal, or divide the block calls between two networks.

18.1.3 CALL Coil/Box

You can call functions and system functions using the CALL coil/box. It is a requirement that the called blocks have no block parameters. You can use the CALL coil/box if a block is too long or not clear enough for you by simply "breaking down" the block into sections and calling the sections one after the other. One single CALL coil or box is permitted per network.

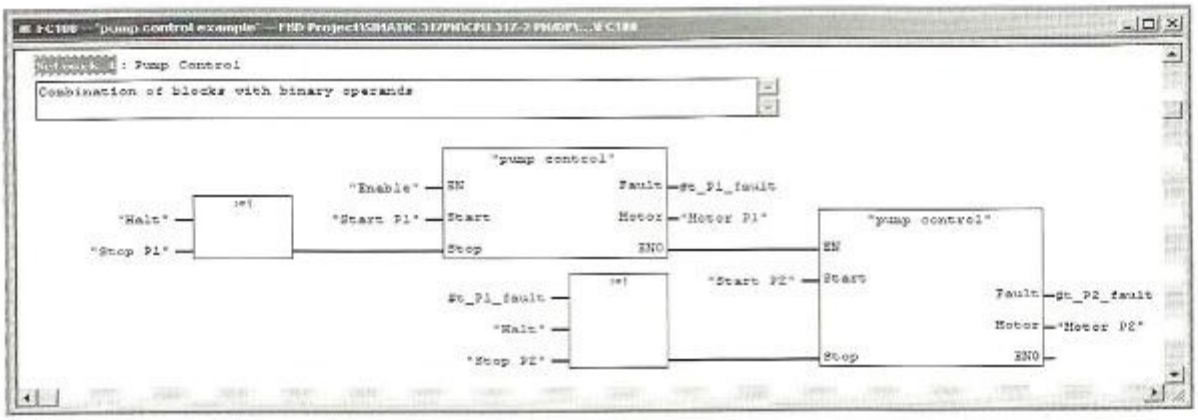


Figure 18.3 Block Calls in Sequence (FBD)

LAD: If the CALL coil is connected directly to the left power rail, the call is always executed (unconditional call). If there is a logic operation preceding the CALL coil, the call is only executed if the preceding logic operation is fulfilled, that is, if power flows into the CALL coil. If the preceding logic operation is not fulfilled, the call is not executed and the next network is processed.

FBD: If there is a logic operation preceding the CALL box, the call is executed only when the preceding logic operation is fulfilled, that is, when RLO = "1" is present at the CALL box. If the preceding logic operation is not fulfilled, the call is not executed and the next network is immediately processed.

When a block change is made, status bit OS is reset; status bits CC0, CC1 and OV are not affected.

MCR dependency is deactivated when a block is called. The MCR is disabled in the called block regardless of whether the MCR was enabled or disabled prior to the block call. When a block is exited, MCR dependency assumes the same setting it had prior to the block call.

Calling a block with the CALL coil/box saves the data block registers in the B stack; the block end restores their contents when the called block is exited. The global data block current prior to the block call and the instance data block are also open following the block call. If no data block was open prior to the block call (for example, no instance data block in OB 1), no data block is open following the block call either, regardless of which data blocks may be open in the called block.

18.1.4 Block End Function

You can terminate processing in a block prematurely with the block end function RET.

Conditional block end

LAD representation:	—(RET)—
FBD representation:	—[RET]

The block end function is represented as a coil or box requiring a preceding logic operation. The RET coil/box must be alone in a network.

If the preceding logic operation is fulfilled, the block is exited. A return jump is made to the previously processed block in which the block call took place. If an organization block is terminated, the CPU returns to the system program.

If the preceding logic operation is not fulfilled, the next network in the block is processed.

IF preceding logic operation == "1"	
THEN	ELSE
The block is exited	The next network is processed
BR := "1"	BR := "0"

The RET coil/box simultaneously stores the RLO (whether power flows or not) in binary result BR, regardless of whether or not the logic operation was fulfilled. The binary result is decisive for controlling the ENO output at the call box (see also Chapter 15, "Status Bits").

18.1.5 Temporary Local Data

You use the temporary local data to buffer results arising during processing of a block. Temporary local data are available only during block processing; once a block has been processed, your buffered data are lost.

Temporary local data are operands which lie in the local data stack (L stack) in system memory. The CPU's operating system makes the temporary local data for a code block available when that code block is called. When a block is called, the values in the L stack are virtually coincidental. In order to be able to make sensible use of the local data, you must first write them prior to reading. When the block is terminated, the L stack is assigned to the next block called.

The number of temporary local data bytes a block requires is in the block header. Reading the header tells the operating system how many bytes have to be reserved in the L stack when the block is called. You, too, can tell from the entry in the block header just how many local data bytes the block requires (using the Editor, with the block open, by invoking FILE → PROPERTIES or in the SIMATIC Manager with EDIT → OBJECT PROPERTIES, in each case on tab "General – Part 2").

Declaring temporary local data

You declare the temporary local data in the declaration section of the code block:

- ▷ under "temp" in the case of incremental programming, or
- ▷ between VAR_TEMP and END_VAR in the case of source-oriented programming.

Figure 18.5 shows an example for the declaration of temporary local data. The variable *difference* lies in the temporary local data and is of data type INT; the variable *buffer* is of data type REAL.

The temporary local data are stored in the L stack order of their declaration in accordance with their data type.

Symbolic addressing of temporary local data

You reference temporary local data using their symbolic names. You assign these names in accordance with the rules for block-local symbols.

All operations allowed for bit memory are also allowed for temporary local data. Please note, however, that a temporary local data bit is not suitable for use as an edge memory bit because it does not retain its signal state outside the relevant block.

You can address the temporary local data for a block only within that block (exception: the temporary local data for the calling block can be accessed via block parameters.)

Size of the L stack

The total size of the L stack is CPU-specific. The available number of temporary local data bytes in a priority class, that is, in the program of an organization block, is also predetermined. On an S7-300, the number of temporary local data bytes is fixed, for example there are 510 bytes per priority class on the CPU 314. On an S7-400, you can specify the number of temporary local data bytes you will need when you initialize the CPU. These bytes must be shared by the blocks called in the relevant organization block as well as by the blocks which they call.

Please note that the Editor also uses temporary local data, for instance for passing block parameters, a fact which goes unnoticed on the programming interface.

Declaration	Name	Data Type	Address
TEMP	SINFO	ARRAY [1..20]	0.0
		BYTE	*1.0
	LByte	ARRAY [1..16]	20.0
		BYTE	*1.0

Figure 18.4
Example of the Declaration of Temporary Local Data in an Organization Block

Start information

When an organization block is called, the CPU operating system passes start information in the temporary local data. This start information comprises 20 bytes for every organization block, and is nearly identical for all OBs. The start information for the various organization blocks is described in detail in Chapters 20, "Main Program", 21, "Interrupt Handling", 22, "Start-up Characteristics" and 23, "Error Handling".

These 20 bytes of information must always be available in each priority class used. If you program a routine for the evaluation of synchronization errors (programming and access errors), you must set aside an additional 20 bytes at least for the start information of these error organization blocks, as these error OBs belong to the same priority class.

You declare the start information for an organization block when you program that block. The information is mandatory. Sample declarations in English can be found on the *Standard Library* under *Organization Blocks*. If you do not need the start information, simply declare the first 20 bytes as something else, for example as an array (as shown in Figure 18.5).

Absolute addressing of temporary local data

Normally, you reference temporary local data by their symbolic names. The use of absolute addresses is the exception. Once you are familiar with the way data are stored in the L stack, you can compute the addresses of the static local data yourself. You will also see the addresses listed in the variable declaration table of the compiled block.

The operand identifier for temporary local data is L; a bit is addressed with L, a byte with LB, a word with LW, and a doubleword with LD.

Example: You want to reserve 16 bytes of temporary local data for absolute addressing, and you want to reference the values in these bytes by both byte and bit. To do this, create an array at the beginning of the local data area so that addressing begins at 0. In an organization block, you would place this array declaration immediately behind the declaration for the start information, in which case addressing would begin at 20.

Data type ANY

A variable in the temporary local data can – although this is an exception – be declared as data type ANY. You can use this feature to modify the ANY pointer at runtime (see Chapter 24.2.5, ""Variable" ANY Pointer").

18.1.6 Static Local Data

Static local data are operands which a function block stores in its instance data block.

Static local data are a function block's "memory". They retain their values until those values are changed by the program, just like data operands in global data blocks.

The number of static local data bytes is limited by the data types of the variables and by the CPU-specific length of a data block.

Declaration	Name	Data Type	Address	Initial Value	Comment
IN	Man_on	BOOL	0.0	FALSE	Input parameter
OUT	Switch_on	BOOL	2.0	FALSE	Output parameter
IN_OUT	Length	INT	4.0	0	I/O parameter
STAT	Total Setpoint	INT DINT	6.0 8.0	0 L#0	Static local data
TEMP	Difference Buffer	INT REAL	0.0 2.0		Temporary local data

Figure 18.5 Example of the Declaration of Local Data in a Function Block

Declaring static local data

You declare static local data in the declaration section of the function block:

- under "stat" in the case of incremental programming or
- between VAR and END_VAR in the case of source-oriented programming.

Figure 18.5 in Chapter 18.1.5, "Temporary Local Data" shows an example of a variable declaration in a function block. The block parameters are declared first, then the static local data, and finally the temporary local data.

The static local data are stored in the instance data block behind the block parameters in the order of their declarations and in accordance with their data types.

Symbolic addressing of static local data

You reference static local data with symbolic names. You assign these names in accordance with the rules for block-local symbols.

All the same operations that can address data operands in global data blocks can also address static local data.

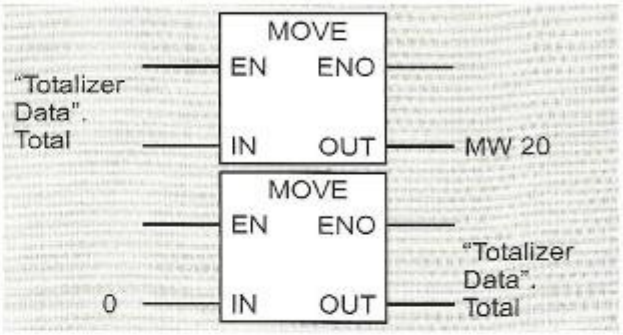
Example: The function block "Totalizer" adds an input value to a value stored in the static local data and then stores the total in the static local data. At the next call, the input value is again added to this total, and so on (Figure 18.6 above).

Total is a variable in the data block "Totalizer-Data", the instance data block for the "Totalizer" function block (you can define the names of all blocks yourself in the Symbol Table, but you must stick to the applicable rules when doing so). The instance data block has the data structure of the function block; in the example, it contains two INT variables with the names *In* and *Total*.

Accessing static data outside the function block

As a rule, the static local data are processed only in the function block itself. Because they are stored in a data block, however, you can access the static local data at any time with *"Data Block Name".Operand Name* just as you would a variable in a global data block.

In our little example, the data block is named *TotalizerData* and the data operand is named *Total*. The applicable access instructions might be as follows:



Local instances

When you call a function block, you normally specify the instance data block for that call. The function block then stores its block parameters and its static local data in this instance data block.

Beginning STEP 7 V2, you can generate "multiple instances", that is, you can call a function block as a local instance in another function block. The static local data (and the block parameters) of the function block called are then a subset of the calling block's static local data. Prerequisite is that both the calling function block and the called function block have block version 2, that is, that both have "multiple instance capability". This allows you to "nest" function block calls up to a depth of eight.

Example (Figure 18.6 below): In the static local data of the function block "Evaluation", you declare a variable *Memory* that corresponds to the function block "Totalizer" and has its data structure. Now you can call the function block "Totalizer" via the variable *Memory*, but without specifying a data block because the data for *Memory* are stored "block-locally" in the static local data (*Memory* is the local instance of the function block "Totalizer").

You access *Memory*'s static local data in the program in function block "Evaluation" the same way you would access structure components, which is by specifying the structure name (*Memory*) and the component name (*Total*).

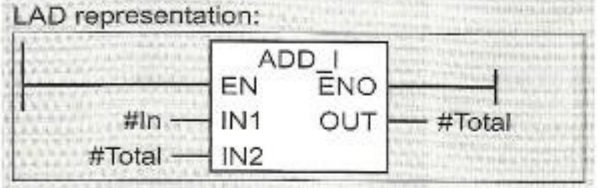
FB "Totalizer"

Address	Declaration	Name	Type
+ 0.0	in	In	INT
+ 2.0	stat	Total	INT

DB "TotalizerData"

Address	Declaration	Name	Type
+ 0.0	in	In	INT
+ 2.0	stat	Total	INT

LAD representation:



FBD representation:

In the Data view, the data block shows all individual variables so that the variables of a local instance appear with their full names.

Simultaneously, you see the corresponding absolute addresses.

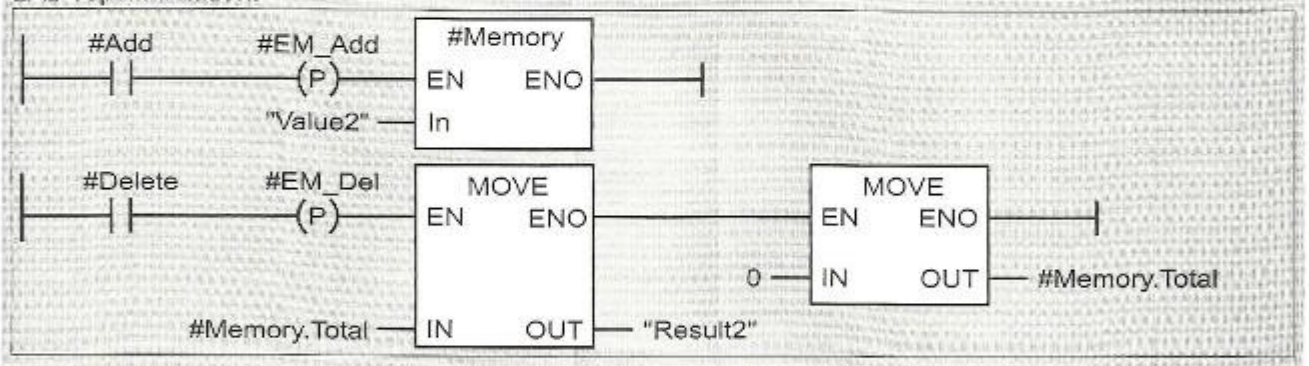
FB "Evaluation"

Address	Declaration	Name	Type
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat	Memory	Totalizer

DB "EvaluationData"

Address	Declaration	Name	Type
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat:in	Memory.In	INT
6.0	stat	Memory.Total	INT

LAD representation:



FBD representation:

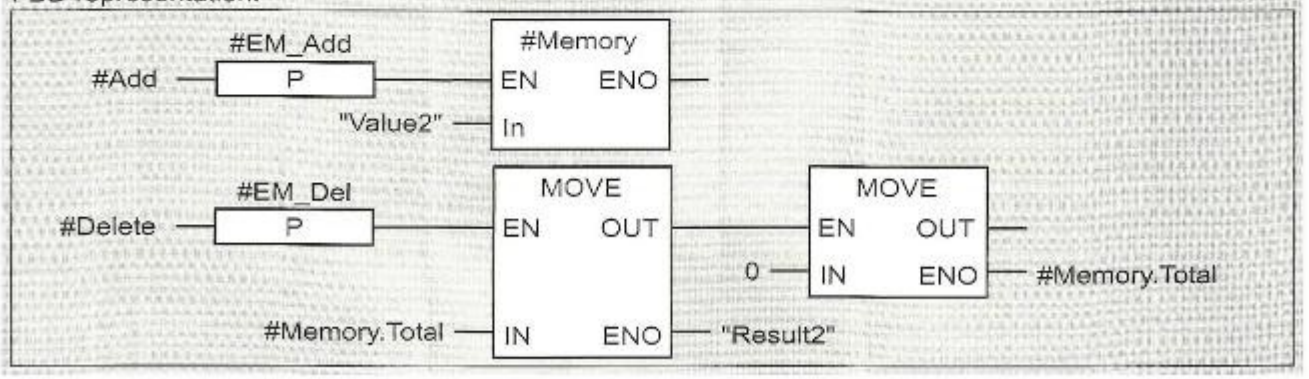


Figure 18.6 Example of Static Local Data and Local Instances

The instance data block "EvaluationData" thus contains the variables *Memory.In* and *Memory.Total*, which you can also address as global variables, for example as *"Evaluation-Data".Memory.Total*.

You can find this example of the use of a local instance in function blocks FB 6, FB 7 and FB 8 in program "Program Flow Control" in the libraries "LAD_Book" and "FBD_Book", that you can download from the publisher's Website (see page 8). The "Feed" example in Chapter 19.5.3, "Feed Example" contains additional usage of local instances.

Absolute addressing of static local data

As a rule, the static local data are addressed symbolically. Absolute addressing is the exception. Within a function block, the instance data block is opened via the DI register. Operands in this data block, and this includes the static local data as well as the block parameters, therefore have DI as operand identifier. A bit is addressed with DIX, a byte with DIB, a word with DIW, and a doubleword with DID.

Once you know just how data are stored in a data block, you can compute the absolute addresses of the static local data yourself. You can also find the addresses in the compiled block in the variable declaration table. But be very careful! *These addresses are addresses that are relative to the start of the instance.* They apply only when you call the function block with a data block. If you call the function block as local instance, the local instance's local data are located right in the middle of the calling function block's instance data block. You can view the absolute addresses, for example, in the compiled instance data block which contains all local instances. Select VIEW → DATA VIEW when you want to read up on the addresses of individual local data operands.

Using our example as a basis, the variable Total in the function block "Totalizer" could be addressed with DIW 2 if function block "Totalizer" were called with a data block (also see the operands in the data block "TotalizerData"), and with DIW 6 if function block "Totalizer" were called as local instance in function block "Evaluation" (also see the operands in data block "EvaluationData").

18.2 Block Functions for Data Blocks

You store your program data in the data blocks. In principle, you can also use the bit memory area for storing data; however, with the data blocks, you have significantly more possibilities with regard to data volume, data structuring and data types. This chapter shows you

- how to work with data operands,
- how to call data blocks and
- how to create, delete and test data blocks at runtime.

You can use data blocks in two versions: as *global data blocks* that are not assigned to any code block, and as *instance data blocks* that are assigned to a function block. The data in the global data blocks are "free" data that every code block can make use of. You yourself determine their volume and structure directly through programming the global data block. An instance data block contains only the data with which the associated function block works; this function block then also determines the structure and storage location of the data in "its" instance data block.

The number and length of data blocks are CPU-specific. The numbering of the data block begins at 1; there is no data block DB 0. You can use each data block either as a global data block or as an instance data block.

You must first create ("set up") the data blocks you use in your program, either by programming, such as code blocks, or at runtime using the system function SFC 22 CREAT_DB.

18.2.1 Two Data Block Registers

Each S7-CPU has two data block registers. These registers contain the numbers of the current data blocks; these are the data blocks with whose operands processing is currently taking place. Before accessing a data block operand, you must open the data block containing the operand. If you use fully-addressed access to data operands (with specification of the data block, see below), you need not be concerned with opening the data blocks or with the contents of the data block registers. The Editor gen-

erates the necessary instructions from your specifications.

The Program Editor uses the first data block register preferably for accessing global data blocks and the second data block register for accessing instance data blocks. For this reason, these registers are given the names "Global data block register" (DB register for short) and "Instance Data Block Register" (DI register for short). The handling of the registers by the CPU is absolutely identical. Each data block can be opened via one of the two registers (or also via both simultaneously).

To come to the essential point first: You can affect only the DB register with LAD or FBD. If you open a data block with the OPN coil/box (see below), you always open it via the DB register. In LAD and FBD, the instance data block is always opened via the DI register; this happens with the call box when the block is called.

When you load a data word, you must specify which of the two possible open data blocks contains the data word. If the data block has been opened via the DB register, the data word is called DBW; if the data word is in the data block opened via the DI register, it is called DIW. The other data operands are named accordingly (Table 18.1).

Table 18.1 Data Operands

Data operand	Located in a data block opened via the	
	DB register	DI register
Data bit	DBX y.x	DIX y.x
Data byte	DBB y	DIB y
Data word	DBW y	DIW y
Data doubleword	DBD y	DID y

x = Bit address, y = Byte address

18.2.2 Accessing Data Operands

You can use the following methods for accessing data operands:

- ▷ Symbolic addressing with full addressing,
- ▷ Absolute addressing with full addressing and
- ▷ Absolute addressing with partial addressing.

Symbolic access to the data operands in global data blocks requires the minimum system knowledge. For absolute access or for using both data block registers, you must observe the notes below.

Symbolic addressing of data operands

It is recommended that you address data operands symbolically whenever possible. Symbolic addressing

- ▷ makes it easier to read and understand the program (if meaningful terms are used as symbols),
- ▷ reduces write errors made during programming (the Program Editor compares the terms used in the Symbol Table and in the program; "number-switching errors" such as DBB 156 and DBB 165 that can occur when using absolute addresses cannot occur here) and
- ▷ does not require programming knowledge at the machine code level (which data block has the CPU opened currently?).

Symbolic addressing uses fully-addressed access (data block together with data operand), so that the data operand always has a unique address.

You determine the symbolic address of a data operand in two steps:

1) Assignment of the data block in the Symbol Table
   Data blocks are global data that have unique addresses within a program. In the Symbol Table, you assign a symbol (e.g. Motor1) to the absolute address of the data block (e.g. DB 51).

2) Assignment of the data operands in the data block
   You define the names of the data operands (and the data type) during programming of the data block. The name applies only in the associated block (it is "block-local"). You can also use the same name in another block for another variable.

Fully-addressed access to data operands

In the case of fully-addressed access, you specify the data block together with the data operand. This method of addressing can be symbolic or absolute:

```
"MOTOR1".ACTVAL
DB 51.DBW 20
```

MOTOR1 is the symbolic address that you have assigned to a data block in the Symbol Table. ACTVAL is the data operand you defined when programming the data block. The symbolic name "MOTOR1".ACTVAL is just as unique a specification of the data operand as the specification DB 51.DBW 20.

Fully-addressed data access is only possible in conjunction with the global data block register (DB register). In the case of fully-addressed data operands, the Program Editor first opens the data block via the DB register and then accesses the data operand.

You can use fully-addressed access with all functions permissible for the data type of the addressed data operand. So with block parameters, for example, you can also specify a fully-addressed data operand as actual parameter.

Absolute addressing of data operands

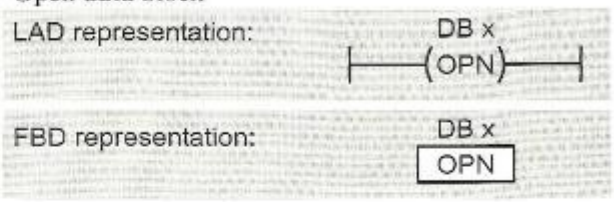
For absolute addressing of data operands, you must know the addresses at which the Program Editor places the data operands when setting up. You can find out the addresses by outputting them after programming and compiling the data block. You will then see from the address column the absolute address at which the relevant variable begins. This procedure is suitable for all data blocks, both those you use as global data blocks as well as those you use as instance data blocks (for local instances see Chapter 18.2.4, "Special Points in Data Addressing"). In this way, you can also see where the Program Editor stores the block parameters and the static local data for function blocks.

Data operands are addressed byte by byte like, for example, bit memory; the functions used on the data operands are also the same as those used with bit memory.

18.2.3 Opening a Data Block

You use the OPN coil (LAD) or OPN box (FBD) to open a data block via the DB register.

Open data block



The OPN coil/box is connected directly to the left power rail and stands alone in a rung. The specified data block is always opened via the DB register. In LAD or FBD, it is not possible to open a data block via the Dl register with OPN coil/box. (The Dl register uses the call box to open the current instance data block.)

In the case of incremental programming, you will find the OPN coil/box in the Program Elements Catalog under "DB Call".

The open data block must be in the work memory at runtime.

In the networks following the OPN coil/box, you can use partial addressing to access only those data operands that are located in the open data block. If you want to copy from one data block to another, you can, for example go into the temporary local data via an intermediate buffer or (better) use fully-addressed data operands. Please note: A fully-addressed data operand overwrites the DB register with "its" data block.

Example: The value of data word DBW 10 of data block DB 13 is to be transferred to data word DBW 16 of data block DB 14. With the MOVE box, you can only copy partially-addressed data operands within the currently open data block. For copying between data blocks, you require an intermediate buffer, such as a local data word (see Figure 18.7). It is better to use full addressing.

When you open a data block, it remains "valid" until another data block is opened. Under certain circumstances, you may not be able to see this via the Program Editor, for example if a block is changed with the call box (see Chapter 18.2.4, "Special Points in Data Addressing").

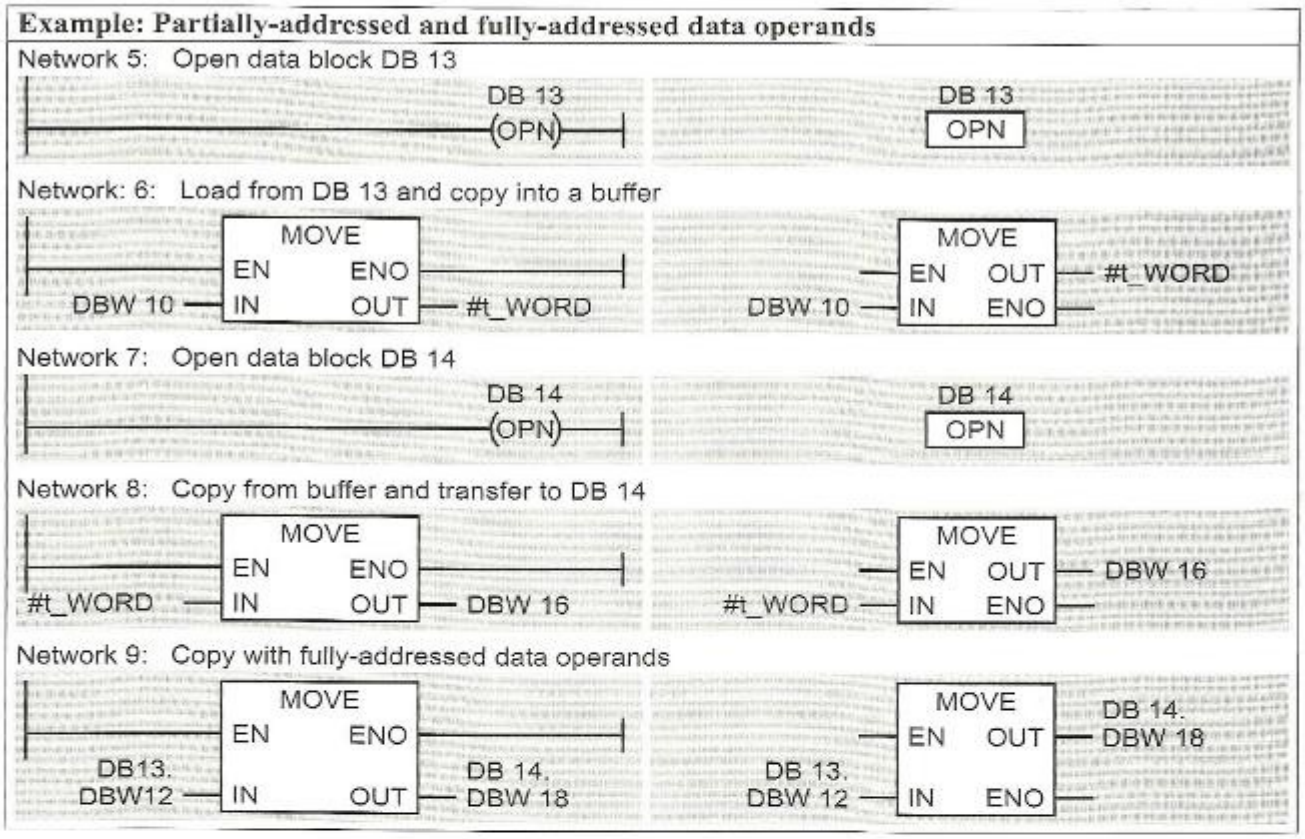


Figure 18.7 Example of Partially- and Fully-Addressed Data Operands

In the case of a block change with the CALL coil/box, the contents of the data block registers are retained. On returning to the calling block, the block change restores the old contents of the registers.

18.2.4 Special Points in Data Addressing

Changes in the contents of the DB registers

With the following functions, the Program Editor generates additional instructions which can alter the contents of the two DB registers:

Full addressing of data operands

Each time data operands are fully addressed, the Program Editor first opens the data block and then accesses the data operand. The DB register is overwritten each time. This also applies to supplying block parameters with fully-addressed data operands.

Accessing block parameters

Access to the following block parameters changes the contents of the DB register: All block parameters of complex data type in the case of functions, and in/out parameters of complex data type in the case of function blocks.

Calling a function block with the call box

Before the actual function block call, the call box stores the number of the current instance data block in the DB register (by swapping the data block registers) and opens the instance data block for the called function block. In this way, the instance data block associated with a called function block is always open. After the actual block call, the call box swaps data block registers again, so that the current instance data block is available again in the calling function block. In this way, the call box changes the contents of the DB register.

DI register in function blocks

In function blocks, the DI register always contains the number of the current instance data block. All access to block parameters or static local data is over the DI register. The address of the local data given in the Declaration Table of the function block applies only when you open the function block with an instance data block, in which case the data operands begin at byte zero.

When you call a function block as local instance, its data are "in the middle" of the calling data block's instance data block. They contain the absolute address of the local data when you output the instance data block in data view format. Each individual variable is then displayed with name and address, including the variables of the local instance called.

When you program a function block that you also want to have available later as local instance, you do not yet know the absolute address of the variable at the time of programming. In this case – just as the Program Editor does for symbolic programming – the contents of address register AR2 are added to the variable address. However, this is possible only in the STL programming language.

Changing the contents of data blocks at a later time

In the Properties window of the offline object container *Blocks* in the "Blocks" register, you can specify whether the absolute address or the symbolic address of the data operand is to have priority for subsequent displays and saves when there is a change in the contents of the data blocks for code blocks that have already been stored.

The default is "Absolute address has priority" (the same as in previous STEP 7 versions). This default means that when there is a change in the declaration, the absolute address is retained in the program while the symbolic address changes accordingly. If the "Symbolic address has priority" default is chosen, the absolute address changes while the symbolic address is retained.

Example: Assuming that data word DBW 10 in data block DB 1 contains the symbolic address of *ActValue*. In the program, you might address this data word with

```
"Data".ActValue    DB1.DBW 10
```

where "Data" is the symbolic address for data block DB 1. If you would now add an additional data word with the symbolic address *MaxCurrent* immediately in front of data word DBW 10, the result upon the next opening (or storing) of the code block will be as follows:

In the case of *"Absolute address has priority"*:

```
"Data".MaxCurrent DB1.DBW 10
```

In the case of *"Symbolic address has priority"*:

```
"Data".ActValue    DB1.DBW 12
```

As for access to data operands in global data blocks, the same applies as for access to global operands (for instance inputs) for which a symbolic address has been entered in the Symbol Table. You will find detailed information on this subject in Chapter 2.5.5, "Address Priority".

Note that this "rewiring" does not take place automatically since the blocks which have already been compiled contain the executable MC7 code of the statements with the absolute address. The change is only carried out in the associated block (with a corresponding message) following opening and saving again.

In order to carry out the modification in the complete block folder, select EDIT → CHECK BLOCK CONSISTENCY with the *Blocks* object selected.

18.3 System Functions for Data Blocks

There are three system functions for handling data blocks. Their parameters are described in Table 18.2.

- ▷ SFC 22 CREAT_DB
  Create data block in work memory
- ▷ SFC 85 CREA_DB
  Create data block in work memory
- ▷ SFC 82 CREA_DBL
  Create data block in load memory
- ▷ SFC 23 DEL_DB
  Delete data block
- ▷ SFC 24 TEST_DB
  Test data block

Table 18.2 SFCs for Handling Data Blocks

SFC	Name	Declaration	Data Type	Assignment, Description
22	LOW_LIMIT	INPUT	WORD	Lowest number of the data block to be created
	UP_LIMIT	INPUT	WORD	Highest number of the data block to be created
	COUNT	INPUT	WORD	Length of the data block in bytes (even number)
	RET_VAL	RETURN	INT	Error information
	DB_NUMBER	OUTPUT	WORD	Number of the created data block
85	LOW_LIMIT	INPUT	WORD	Lowest number of the data block to be created
	UP_LIMIT	INPUT	WORD	Highest number of the data block to be created
	COUNT	INPUT	WORD	Length of the data block in bytes (even number)
	ATTRIB	INPUT	BYTE	Block attribute: B#16#00 Retain B#16#04 Non_Retain
	RET_VAL	RETURN	INT	Error information
	DB_NUMBER	OUTPUT	WORD	Number of the created data block
82	REQ	INPUT	BOOL	Trigger for generation with signal state "1"
	LOW_LIMIT	INPUT	WORD	Lowest number of the data block to be created
	UP_LIMIT	INPUT	WORD	Highest number of the data block to be created
	COUNT	INPUT	WORD	Length of the data block in bytes (even number)
	ATTRIB	INPUT	BYTE	Attributes of created data block
	SRCBLK	INPUT	ANY	Data area in work memory with which the created data block is initialized
	RET_VAL	RETURN	INT	Error information
	BUSY	OUTPUT	BOOL	If TRUE, creation has not yet been finished
	DB_NUM	OUTPUT	WORD	Number of the created data block
23	DB_NUMBER	INPUT	WORD	Number of the data block to be deleted
	RET_VAL	RETURN	INT	Error information
24	DB_NUMBER	INPUT	WORD	Number of the data block to be tested
	RET_VAL	RETURN	INT	Error information
	DB_LENGTH	OUTPUT	WORD	Length of the data block (in bytes)
	WRITE_PROT	OUTPUT	BOOL	TRUE = write-protected

18.3.1 Creating a Data Block in Work Memory

System functions SFC 22 CREAT_DB and SFC 85 CREA_DB create a data block in the work memory. As the data block number, the system function takes the lowest free number in the number band given by the input parameters LOW_LIMIT and UP_LIMIT. The numbers specified at these parameters are included in the number band. If both values are the same, the data block is created with this number. The number of a data block already present in the user program cannot be assigned again, not even if the data block is only present in the load memory.

The output parameter DB_NUMBER supplies the number of the data block actually created. With the input parameter COUNT, you specify the length of the data block to be created. The length corresponds to the number of data bytes and must be an even number.

Creating the data block is not the same as calling it. The current data block is still valid. A data block created with this system function contains random data. For meaningful use, data

must first be written to a data block created in this way before the data can be read.

The data blocks created with the SFCs 22 CREAT_DB and 85 CREA_DB are only present in work memory. If a CPU differentiates between retentive and non-retentive work memory, the SFC 22 CREAT_DB creates a retentive data block, and the SFC 85 CREA_DB creates a data block as specified by the ATTRIB parameter. "Retentive" data block means that is its contents are retained in the event of a warm restart/hot restart (see Chapter 22.2.4, "Retentivity").

The system function SFC 85 CREA_DB replaces the SFC 22 CREAT_DB.

A data block created by the SFCs 22 CREAT_DB and SFC 85 CREA_DB does not change the checksum of the user program, not even if it is written or deleted again. If a data block created by an SFC is imported into the offline data management, this has an influence on the checksum.

In the event of an error, a data block is not created, the parameter DB_NUMBER parameter is assigned zero, and an error number is output by RET_VAL.

18.3.2 Creating a Data Block in Load Memory

System function SFC 82 CREA_DBL creates a data block in the load memory. As the data block number, the system function takes the lowest free number in the number band given by the input parameters LOW_LIMIT and UP_LIMIT. The numbers specified at these parameters are included in the number band. If both values are the same, the data block is created with this number. The number of a data block already present in the user program cannot be assigned again, not even if the data block is only present in the work memory.

The output parameter DB_NUM supplies the number of the data block actually created. With the input parameter COUNT, you specify the length of the data block to be created. The length corresponds to the number of data bytes and must be an even number.

The data area assigned as default to the created data block is specified at input parameter SRCBLK. Here you can specify a complete data block, e.g. DB 160 or "Archive 1", a variable from a data block, or an absolute addressed data area as an ANY pointer, e.g. P#DB160.DBX16.0 BYTE 64. The source must be a data area in the work memory.

If the source area is smaller than the target area, the source area is written completely into the target area. The remaining bytes of the target area are filled with zeros. If the source area is larger than the target area, the target area is written completely; the remaining bytes of the source area are ignored.

You can assign the following attributes to the created data block using the ATTRIB parameter:

- Bit 0 = "1"
  The data block has the *Unlinked* attribute. Following transfer to the offline data management and reloading into the CPU, the data block is still only present in the load memory.
- Bit 1 = "1"
  The data block has the *DB is write-protected in the PLC* attribute. You can only read the values of this data block.
- Bit 2 = "1"
  The data block has the *Non_Retain* attribute.

The other bits are not used at the moment. You can find further information on the block attributes in Chapter 3.2.3, "Block Properties"

System function SFC 82 CREA_DBL operates in asynchronous mode: you trigger the creation process with signal state "1" at input parameter REQ. You may only access the read and written data areas again when the output parameter BUSY has returned to the signal state "0".

Creating does not call the associated data block. The current data block is still the valid one.

A data block is not created in the event of an error, the output parameters are assigned undefined values, and an error message is output by the function value.

18.3.3 Deleting a Data Block

System function SFC 23 DEL_DB deletes the data block in RAM (work and load memory) whose number is specified in the input parameter DB_NUMBER. The data block must not be open at the time, as otherwise the CPU will go to STOP.

Data blocks created with the keyword *Unlinked* and data blocks on an Flash EPROM memory card cannot be deleted with system function SFC 23.

In the event of an error, the data block is not deleted and an error number is returned as function value.

18.3.4 Testing a Data Block

System function SFC 24 TEST_DB provides information on a data block whose number you specify at input parameter DB_NUMBER. Output parameter DB_LENGTH contains the number of existing bytes, and output parameter WRITE_PROT indicates whether the data block is write-protected.

If the tested data block is only in load memory, this is indicated as an error by RET_VAL; the DB_LENGTH and WRITE_PROT parameters nevertheless have the correct assignments.

If the specified data block is not present in the CPU's work memory, RET_VAL is returned with W#16#80B1.

19 Block Parameters

This chapter describes how to use block parameters. You will learn

- how to declare block parameters,
- how to work with block parameters,
- how to initialize block parameters and
- how to 'forward' block parameters.

Block parameters represent the transfer interface between the calling and the called block. All functions and function blocks can be provided with block parameters.

19.1 Block Parameters in General

19.1.1 Defining the Block Parameters

Block parameters make it possible to parameterize the processing instruction in a block, the block function. Example: You want to write a block as an adder that you can use in your program several times with different variables. You transfer the variables as block parameters; in our example, three input parameters and one output parameter (Figure 19.1). Since the adder need not store any values internally, a function is suitable as the block type.

You define a block parameter as an input parameter if you only check or load its value in the block program. If you only describe a block parameter (assign, set, reset, transfer), you use an output parameter. You must always use an in/out parameter if a block parameter is to be both checked and overwritten. The Program Editor does not check the use of the block parameters.

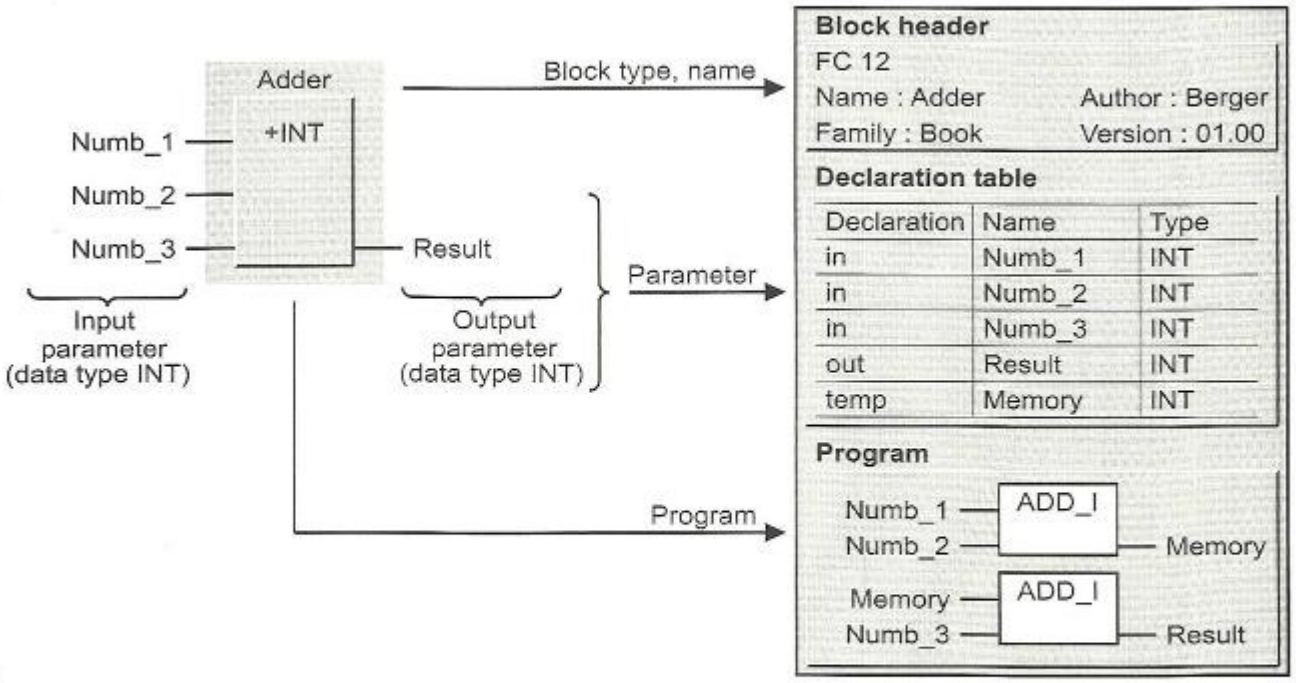


Figure 19.1 Example of Block Parameters

19.1.2 Processing the Block Parameters

In the adder program, the names of the block parameters stand as place holders for the latest actual variables. You use the block parameters in the same way as symbolically addressed variables; in the program, they are called "formal parameters".

You can call the "Adder" function several times in your program. With each call, you transfer other values to the adder in the block parameters (Figure 19.2). The values can be constants, operands or variables; they are called "actual parameters".

At runtime, the CPU replaces the formal parameters with the actual parameters. The first call in the example adds the contents of memory words MW 30, MW 32 and MW 34 and stores the result in memory word MW 40. The same block with the actual parameters of the second call adds data words DBW 30, DBW 32 and DBW 34 of data block DB 13 and stores the result in data word DBW 40 of data block DB 14.

19.1.3 Declaration of the Block Parameters

You define the block parameters in the declaration section of the block when you program the block. Figure 19.3 shows the declaration tables of a function FC and a function block FB. In addition to the block parameters (IN, OUT, IN_OUT), you also declare the temporary local data (TEMP), the function value (RETURN) for functions FC, and the static local data (STAT) for function blocks. Only the temporary local data (TEMP) exists for organization blocks OB which are neither called in the user program nor possess an instance data block.

Default values are optional and are only possible for function blocks if a block parameter is stored as a value. This applies to all block parameters of elementary data type and to input and output parameters of complex data type. A parameter comment can also be given.

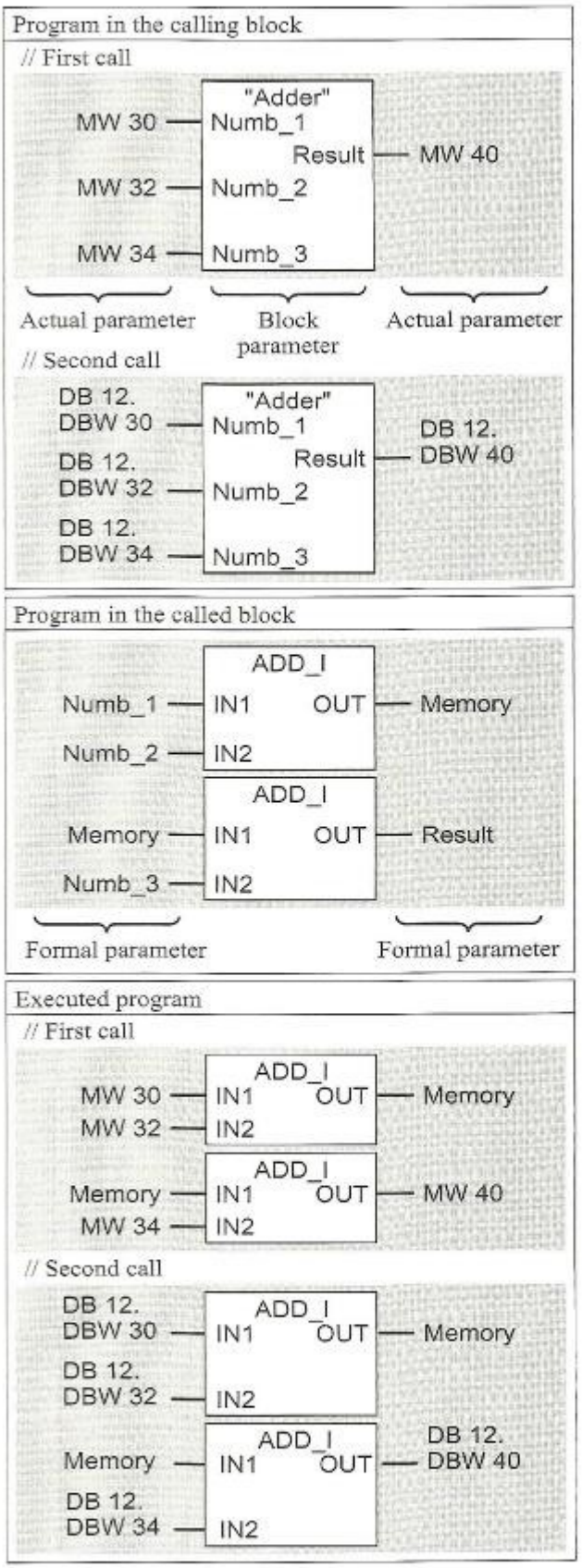


Figure 19.2 Block Call with Block Parameters

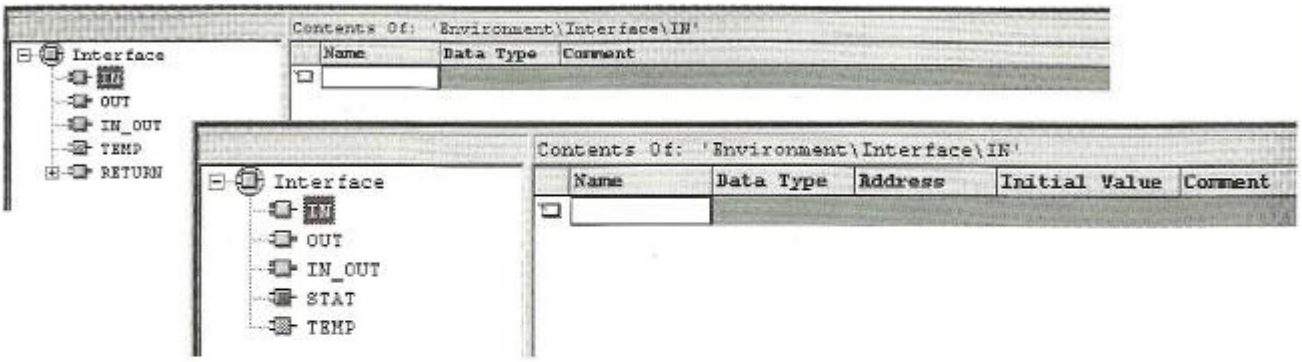


Figure 19.3 Empty Declaration Tables for Functions FC and Function Blocks FB

The *block parameter name* can be up to 24 characters in length. It must consist only of letters (without national characters such as the German Umlaut), digits, and the underscore. No distinction is made between upper and lower case. The name must not be a keyword.

For the *data type* of a block parameter, all elementary, complex and user-defined data types are permissible as well as the parameter types (see Chapter 3.5, "Variables, Constants and Data Types").

STEP 7 stores the names of the block parameters in the non-executable section of the blocks on the programming device's storage medium. The work memory of the CPU (in the compiled block) contains only the declaration types and the data types. For this reason, program changes made to blocks online in the CPU must always be updated on the programming device's data medium in order to retain the original names. If the update is not made, or if blocks are transferred from the CPU to the programming device, the non-executable-block sections are overwritten or deleted. The Program Editor then generates replacement symbols for display or printout.

19.1.4 Declaration of the Function Value

In functions, the function value is a specially treated output parameter. It has the name RET_VAL (or ret_val) and is defined as the first output parameter. You declare the function value by specifying the data type and possibly a comment in the declaration table under RETURN and the name RET_VAL.

As data type of the function value, all elementary data types as well as the data types DATE_AND_TIME, STRING, POINTER, ANY and user-defined data types UDT are permitted. The data types ARRAY and STRUCT are not allowed.

As the first output parameter, the function value has no special role to play in the LAD or FBD programming language. It only gains significance in the SCL programming language, where you can use the block type FUNCTION as a "genuine" function. Here, a function FC can stand in place of an operand in a printout; the function value then represents the value of this function.

19.1.5 Initializing Block Parameters

When calling a block with block parameters, you initialize the block parameters with actual parameters. These can be constants, operands with absolute addresses, fully-addressed data operands or symbolically addressed variables. The actual parameter must be of the same data type as the block parameter.

You must initialize all of a function's block parameters at every call. In the case of function blocks, initialization of individual block parameters or all block parameters is optional.

19.2 Formal Parameters

In this chapter, you will learn how to access the block parameters within a block. Table 19.1 shows that it is possible to access block parameters of elementary data type, components of an array or structure, and timers and counters without restriction. Chapter 19.4, ""Forwarding" Block Parameters".

Access to parameters of complex data type and of parameter types POINTER and ANY is not supported by LAD or FBD. However, you can initialize acquired blocks or system blocks that have such parameters with the relevant variable. You can find examples of this in the libraries "LAD_Book" and "FBD_Book" under the program "Data Types" that you can download from the publisher's Website (see page 8).

Block parameters with data type BOOL

Block parameters of data type BOOL can be individual binary variables or binary components of arrays and structures. You can check input parameters and in/out parameters with contacts or with binary box inputs, and you can influence output parameters and in/out parameters with binary box outputs. After the CPU has used the actual parameter specified as block parameter, it processes the functions as shown in the relevant chapters.

Block parameters of digital data type

Block parameters of digital data type occupy 8, 16 or 32 bits (all elementary data types except BOOL). They can be individual digital variables or digital components of arrays and structures. You apply input and in/out parameters to digital box inputs, and you write output and in/out parameters to digital box outputs.

You exchange values between block parameters of different types or different sizes with the MOVE box as described in Chapter 6, "Move Functions".

Table 19.1 Accessing Block Parameters (General)

Data Types	Permitted for			Access in Block Possible
	IN	I_O	OUT	
Elementary data types				
BOOL	x	x	x	yes
BYTE, WORD, DWORD, CHAR, INT, DINT, REAL, S5TIME, TIME, TOD, DATE	x	x	x	yes
Complex data types				
DT, STRING	x	x	x	no
ARRAY, STRUCT				
Individual binary components	x	x	x	yes
Individual digital components	x	x	x	yes
Complete variables	x	x	x	no
Parameter types				
TIMER	x	-	-	yes
COUNTER	x	-	-	yes
BLOCK_FC, BLOCK_FB	x	-	-	yes
BLOCK_DB	x	-	-	yes
BLOCK_SDB	x	-	-	no
POINTER, ANY	x	x	x[1)]	no

[1)] FC functions only

Block parameters of data type DT and STRING

Direct access to block parameters of type DT and STRING is not possible. In function blocks, you can "pass" input and output parameters of data type DT and STRING to parameters of called block.

Block parameters of data type ARRAY and STRUCT

Direct access to block parameters of type ARRAY and STRUCT is possible on a component-by-component basis, that is, you can access individual binary or digital components with the relevant operations.

Access to the complete variable (entire array or entire structure) is not possible and neither is access to individual components of complex or user-defined data type. In function blocks, you can "pass" input and output parameters of data type ARRAY and STRUCT to parameters of called blocks.

Block parameters of user-defined data type

You handle block parameters of user-defined data type in the same way as block parameters of data type STRUCT.

Direct access to block parameters of data type UDT is possible on a component-by-component basis, that is, you can access individual binary or digital components with the relevant operations.

Access to the complete variable is not possible and neither is access to individual components of complex or user-defined data type. In function blocks, you can "pass on" input and output parameters of data type UDT to parameters of called blocks.

Block parameters of data type TIMER

You can use block parameters of data type TIMER with all functions as described in Chapter 7, "Timers". When a timer is started, the time value can also be a block parameter of data type S5TIME.

Block parameters of data type COUNTER

You can use block parameters of data type COUNTER with all functions as described in Chapter 8, "Counters". When setting a counter, the counter value can also be a block parameter of data type WORD.

Block parameters of data type BLOCK_DB

You can transfer a data block via a block parameter of data type BLOCK_DB. Call this data block with the OPN coil/box by labeling the OPN coil/box with the formal parameters. When opening a data block via a block parameter, the CPU always uses the global data block register (DB register).

Block parameters of data type BLOCK_FC

You can transfer an FC function via a block parameter of data type BLOCK_FC. Call this function with the CALL coil/box. You can use the CALL coil/box with a formal parameter and with or without any preceding logic operation if you are currently programming a function block. If you use the CALL coil/box with formal parameters in a function, a preceding logic operation is not permissible (absolute call only).

An FC function transferred via a block parameter must not have any block parameters.

Block parameters of data type BLOCK_FB

You can transfer an FB function block via a block parameter of data type BLOCK_FB. Direct access to block parameters of data type BLOCK_FB is not possible with LAD or FBD functions.

An FB function block transferred via a block parameter must not have any block parameters.

Block parameters of data type POINTER and ANY

Direct access to block parameters of data type POINTER and ANY is not possible with LAD or FBD functions.

19.3 Actual Parameters

When you call a block, you initialize its block parameters with constants, operands or variables with which it is to operate. These are the actual parameters. If you call the block often in your program, you usually use different actual parameters each time.

An actual parameter must agree in data type with the block parameter: You can only apply a binary actual parameter (for example, a memory bit) to a block parameter of type BOOL; you can only initialize a block parameter of type ARRAY with an identically dimensioned array variable. Table 19.2 gives an overview of which operands you can use as actual parameters with which data type.

When calling functions, you must initialize all block parameters with actual parameters.

When calling function blocks, it is not a mandatory requirement that you initialize block parameters with actual parameters. If you make no specification at a block parameter, the (old) values stored in the instance data block are used as the actual parameters. These can be, for example, the default values or the values of actual parameters from an earlier call. In/out parameters of complex data type cannot be assigned default values, and neither can parameter types. You must provide these block parameters with actual parameters at least at the first call.

You can also use direct access to access the block parameters for the function block. Since they are located in a data block, you can handle the block parameters like data operands. Example: A function block with the instance data block *"Station_1"* controls a binary output parameter with the name *"Up"*. Following processing in the function block (after its call), you can check the parameter under the symbolic address *"Station_1".Up,* without having initialized the output parameter.

Initializing block parameters with elementary data types

The actual parameters listed in Table 19.3 are permissible as actual parameters of elementary data type.

You can assign either absolute or symbolic addresses to input, output and bit memory operands. Input operands should be used only for input parameters and output operands for output parameters (however, this is not mandatory). Bit memory operands are suitable for all declaration types. You must apply peripheral inputs only to input parameters and peripheral outputs only to output parameters.

You use the Boolean constants TRUE and FALSE in LAD and FBD to set defaults in the declaration of block parameters, static local data or data operands. Initializing block parameters with these constants is not permitted. You also cannot create these constants at a contact (LAD) or at a function input (FBD). If you want

Table 19.2 Initialization with Actual Parameters

Data Type of the Block Parameter	Permissible Actual Parameters
Elementary data type	▷ Simple operands, fully-addressed data operands, constants ▷ Components of arrays or structures of elementary data type ▷ Block parameters of the calling block ▷ Components of block parameters of the calling block of elementary data type
Complex data type	▷ Variables or block parameters of the calling block
TIMER, COUNTER and BLOCK_xx	▷ Timers, counters and blocks
POINTER	▷ Simple operands, fully-addressed data operands ▷ Range pointer or DB pointer
ANY	▷ Variables of any data type ▷ ANY pointer

Table 19.3 Actual Parameters of Elementary Data Type

Operands	Permissible with			Binary operand or symbolic name	Digital operand or symbolic name
	IN	I_O	OUT		
Inputs (process image)	x	x	x	I y.x	EB y, EW y, ED y
Outputs (process image)	x	x	x	Q y.x	AB y, AW y, AD y
Bit memory	x	x	x	M y.x	MB y, MW y, MD y
Peripheral inputs	x	-	-	-	PEB y, PEW y, PED y
Peripheral outputs	-	-	x	-	PAB y, PAW y, PAD y
Global data					
Partial addressing	x	x	x	-	DBB y, DBW y, DBD y
Full addressing	x	x	x	DB z.DBX y.x	DB z.DBB y, etc.
Temporary local data	x	x	x	L y.x	LB y, LW y, LD y
Static local data	x	x	x	DIX y.x	DIB y, DIW y, DID y
Constants	x	-	-	-	All digital constants
Components of ARRAY or STRUCT	x	x	x	Complete component name	Complete component name

x = Bit number, y = Byte address, z = Data block number

to initialize block parameters with TRUE or FALSE, you require a variable that is permanently assigned signal state "1" or "0" and that you use instead of the Boolean constants. Bit memories are obvious candidates for variables in this case. The following example shows you how to set a bit memory permanently to signal state "1" or "0" at startup, see Figure 19.4.

Part-addressed data bits are not permissible as actual parameters in LAD and FBD. The reason for this is that the Program Editor transfers the signal state of every binary actual operand to a temporary local data bit prior to the actual block call. For this reason, it is possible in LAD and FBD to initialize a binary block input with a logic operation. During copying before the block call, the assignment of the data block to the data bit is lost. Even if you open the "correct" data block in the program of the called block prior to accessing the block parameter occupied by a partially addressed data bit, only the previously created copy of the signal state will be scanned.

The same applies to binary in/out parameters and output parameters: here too, assignment with part-addressed data bits is not permissible (the Program Editor will not reject such an assignment but in most cases it will result in errored functions).

When using part-addressed data operands at digital parameters, you must note that when accessing the block parameters (in the called block), the currently opened data block is also the "correct" data block. Since the Editor may in certain circumstances, such as during a block call or if the block parameters have previously been accessed, not visibly change the data block, the use of partially addressed digital data operands is not recommended. Use only fully addressed data operands for this reason.

Temporary local data are usually symbolically addressed. They are located in the L stack of the calling block and are declared in the calling block.

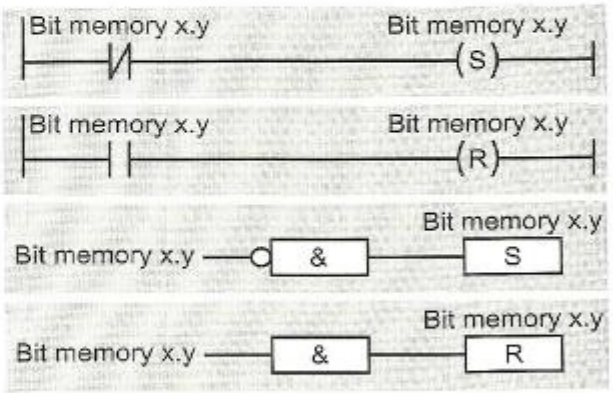


Figure 19.4 Initialization with Fixed Signal State

If the calling block is a function block, you can also use its static local data as actual parameters (see Chapter 19.4, ""Forwarding" Block Parameters"). Static data are usually symbolically addressed; if you would like to assign absolute addresses via DI operands, please note the information in Chapter 18.2.4, "Special Points in Data Addressing".

With a block parameter of type BOOL, you can apply the constant TRUE (signal state "1") or FALSE (signal state "0"), and with block parameters of digital data type, you can apply all constants corresponding to the data type. Initialization with constants is permissible only for input parameters.

You can also initialize a block parameter of elementary data type with components of arrays and structures if the component is of the same data type as the block parameter.

Initializing block parameters of complex data type

Every block parameter can be of the complex type. Variables of the same type can be used as actual operands.

For initializing block parameters of type DT or STRING, individual variables or components of arrays or structures of the same data type are permissible.

STRING variables can be of variable length. If you have specified the STRING length when declaring an input or output parameter for a function block, the Editor reserves the specified space in the instance data block; if you have not specified any length, 256 bytes are reserved for the STRING variable. The maximum length of the STRING variable, which you specify in the declaration, must be the same for the actual parameter and the formal parameter. Exception: With FC functions, you specify either no length or the standard length of 254 bytes when declaring a STRING variable; here, you can use STRING variables of all lengths as actual parameters.

For initializing block parameters of type ARRAY or STRUCT, variables with exactly the same structure as the block parameters are permissible.

Initializing block parameters of user-defined data type

For complex or extensive data structures, the use of user-defined data types (UDTs) is recommended. First, you define the UDT and then you use it, for example, to generate the variable in the data block or to declare the block parameter. You can then use the variable when initializing the block parameter. Here, too, the actual parameter (the variable) must be of the same data type (the same UDT) as the block parameter.

A complete data block with the same UDT type as the block parameter is not permissible as actual parameter.

Initializing block parameters of type TIMER, COUNTER and BLOCK_xx

You initialize a block parameter of type TIMER with a timer and a block parameter of type COUNTER with a counter. You can apply only blocks without their own parameters to block parameters of parameter types BLOCK_FC and BLOCK_FB. You initialize BLOCK_DBs with a data block.

Block parameters of types TIMER, COUNTER and BLOCK_xx must be input parameters.

Initializing block parameters of type POINTER

Pointers (constants) and operands are permissible for block parameters of parameter type POINTER. These pointers are either range pointers (32-bit pointers) or DB pointers (48-bit pointers). The operands are of elementary data type and can also be fully-addressed data operands.

Output parameters of type POINTER are not permitted for function blocks.

Initializing block parameters of type ANY

Variables of all data types are permissible for block parameters of parameter type ANY. The programming within the called block determines which variables (operands or data types) must be applied to the block parameters, or which variables are feasible. You can also specify a constant in the format of the ANY pointer

Table 19.4 Permissible Combinations when Forwarding Block Parameters

Calling → called	FC calls FC			FB calls FC			FC calls FB			FB calls FB		
Declaration type	E	C	P	E	C	P	E	C	P	E	C	P
Input → Input	x	-	-	x	x	-	x	-	x	x	x	x
Output → Output	x	-	-	x	x	-	x	-	-	x	x	-
In/out → Input	x	-	-	x	-	-	x	-	-	x	-	-
In/out → Output	x	-	-	x	-	-	x	-	-	x	-	-
In/out → In/out	x	-	-	x	-	-	x	-	-	x	-	-

E = Elementary data types
C = Complex data types
P = Parameter types TIMER, COUNTER and BLOCK_xx

"P#[data block.]Operand Datatype Number" and thus define an absolute-addressed area.

Output parameters of type ANY are not permissible for function blocks.

19.4 "Forwarding" Block Parameters

"Forwarding" block parameters is a special form of access and of initializing block parameters. The block parameters of the calling block are "forwarded" to the parameters of the called block. The formal parameter of the calling block then becomes the actual parameter of the called block.

In general, it is also the case here that the actual parameter must be of the same type as the formal parameter (that is, the relevant block parameters must agree in their data types). In addition, you can apply an input parameter of the calling blocks only to an input parameter of the called block, and similarly, an output parameter to an output parameter. You can apply an in/out parameter of the calling block to all declaration types of the called block.

There are restrictions with regard to data types caused by the variations in the storage of block parameter for functions and those for function blocks. Block parameters of elementary data type can be passed on without restriction in accordance with the information in the previous paragraph. Input and output parameters of complex type can only be forwarded if the calling block is a function block.

Block parameters of parameter types TIMER, COUNTER and BLOCK_xx can only be passed on from one input parameter to another if the calling block is a function block. These statements are represented in Table 19.4.

19.5 Examples

19.5.1 Conveyor Belt Example

The example shows the transfer of signal states via block parameters. For this purpose, we use the function of a conveyor belt control system described in Chapter 5, "Memory Functions". The conveyor belt control system is to be programmed in a function block, and all inputs and outputs are to be routed through block parameters so that the function block can be called repeatedly (for several conveyor belts). Figure 19.5 shows the input and output parameters for the function block as well as the static and temporary local data used.

Distributing the parameters is quite simple in this case: All binary operands that were inputs have become input parameters, all outputs have become output parameters, and all memory bits have become static local data. You will also have noticed that the names have been altered slightly because only letters, digits, and the underscore are permitted for block-local variables.

The function block "Conveyor_Belt" is to control two conveyor belts. For this purpose, it will be called twice; the first time with the inputs and outputs of conveyor belt 1 and the second

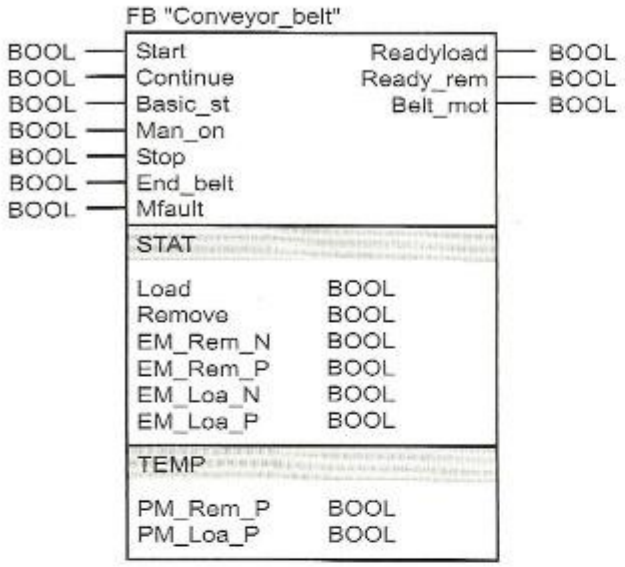


Figure 19.5
Function Block for the Conveyor Belt Example

time with those of conveyor belt 2. For each call, the function block requires an instance data block where it stores the data for the obtaining conveyor belt. The data block for conveyor belt 1 is to be called "BeltData1" and the data block for conveyor belt 2 is to be called "BeltData2".

You can find the executed programming example in the library "LAD_Book" or "FBD_Book" under the program "Conveyor Example" that you can download from the publisher's Website (see page 8). It shows the programming of function block FB 21 with input parameters, output parameters and static local data. You can use any data blocks as instance blocks; in the example, DB 21 is used for "Belt_data1" and DB 22 for "Belt_data2". In the Symbol Table, these data blocks have the data type of the function block (FB 21 in the example, if "Conveyor_belt" is the symbol for FB 21).

When you call the function block, you can use the inputs and outputs from the Symbol Table as actual parameters. In those cases where these global symbols contain special characters, you must place these symbols in quotation marks in the program. The Symbol Table is designed for all three examples in this chapter (Table 19.6 at the end of this chapter).

19.5.2 Parts Counter Example

The example demonstrates the handling of block parameters of elementary data type. The "Parts Counter" example from Chapter 8, "Counters", is the basis of the function. The same function is implemented here as a function block, with all global variables declared either as block parameters or as static local data. The timers and counters are controlled here via individual elements.

Timers and counters are transferred via block parameters of type TIMER and COUNTER. These block parameters must be input parameters. The initial values of the counter (*Quantity*) and the timer (*Dura1* and *Dura2*) can also be transferred as block parameters; the data types of the block parameters correspond here to those of the actual parameters.

The edge memory bits are stored in the static local data and the pulse memory bits are stored in the temporary local data.

You can find the sample program in the library "LAD_Book" or "FBD_Book" under the program "Conveyor Example", that you can download from the publisher's Website (see page 8). It contains function block FB 22 "Parts_counter" and the associated instance data block

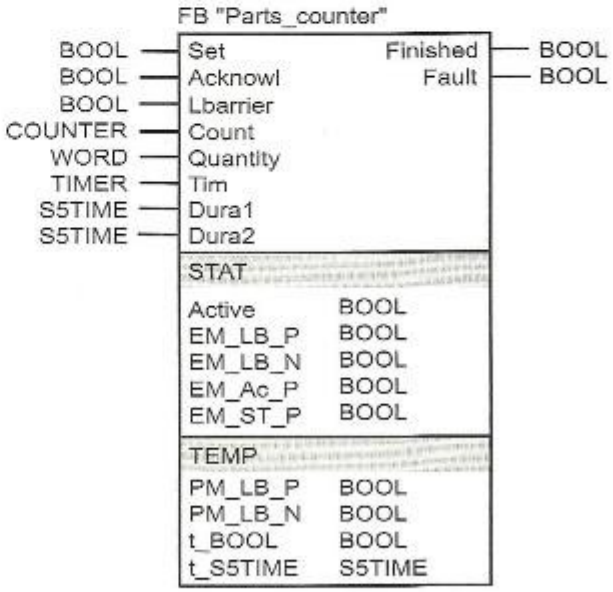


Figure 19.6
Function Block for the Parts Counter Example

"CountDat". You can use the inputs, outputs, timer and counter from the Symbol Table (Table 19.6 in previous example) as actual parameters when calling the function block.

19.5.3 Feed Example

The same functions described in the two previous examples can also be called as local instances. In our example, this means that we program a function block called "Feed" that is to control four conveyor belts and count the conveyed parts. In this function block, FB "Conveyor Belt" is called four times and FB "Parts_counter" is called once. The call does not take place in each case with its own instance data block, but the called FBs are to store their data in the instance data block of function block "Feed".

Figure 19.7 shows how the individual conveyor belt controls are interconnected (FB "Parts_counter" is not represented here). The start signal is connected to the *Start* input of the controller for belt 1, the *ready_rem* output is connected to the *Start* input of belt 2, etc. Finally, the *ready_rem* output of belt 4 is connected to the *Remove* output of "Feed". The same signal sequence leads in the reverse direction from *Remove* over *Continue* and *Readyload* to *Load*.

Belt_mot, Lbarr and */Mfault* are separate conveyor belt signals; *Reset, Startup* and *Stop* control all conveyor belts via *Basic_st, Man_on* and *Stop*.

The following program for function block "Feed" is designed in the same way. The input and output parameters of the function block can be taken from Figure 19.7. In addition, the digital values for the parts counter *Quantity, Dura1* and *Dura2* are designed as input parameters here. We declare the data of the individual conveyor belt controls and the data of the parts counter in the static local data in exactly the same way as for a user-defined data type, i.e. with name and data type. The variable "Belt1" is to receive the data structure of function block "Conveyor Belt", as is variable "Belt2", etc.; the variable "Check" receives the data structure of the function block "Parts_counter".

The program in the function block starts with the initialization of the signals common to all conveyor belts. Here, we make use of the fact that the block parameters of the function blocks called as local instances are static local data in the current block and can be treated as such. The block parameter *Man_on* in the current function block controls the *Man_on* input parameters of all four conveyor belt controls with a single assignment. We proceed in the same way with the signals *Stop* and *Reset*. And now the conveyor belt controls are initialized with the common signals. (You can, of course, also initialize these input parameters when the function block is called.)

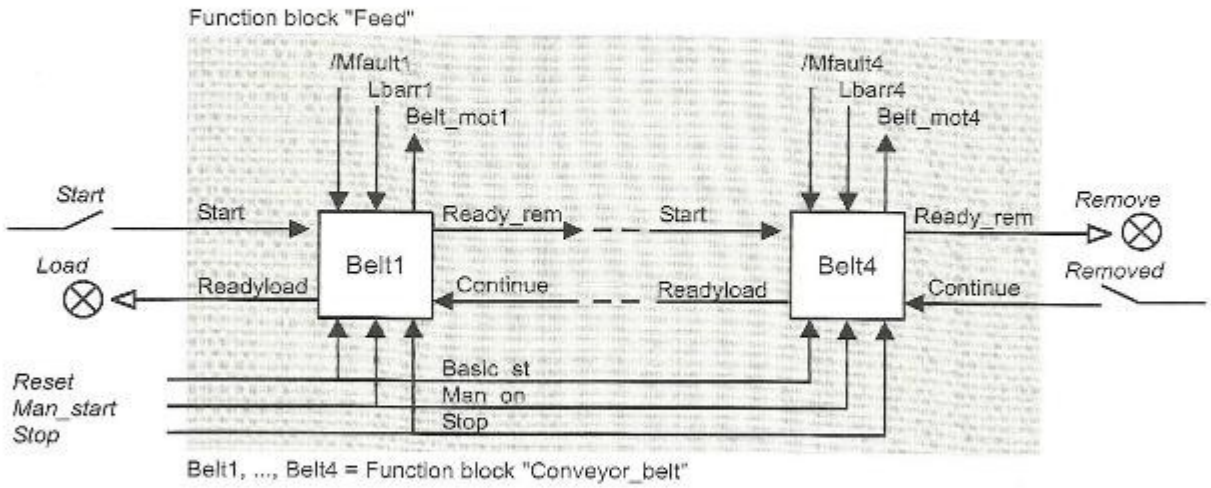


Figure 19.7 Feed Programming Example

Table 19.5 Declaration Section of FB "Feed"

Declaration	Name	Data Type	Address	Initial Value	Comment
IN	Start	BOOL	0.0	FALSE	Start conveyor belts
	Removed	BOOL	0.1	FALSE	Parts have been removed from belt
	Man_start	BOOL	0.2	FALSE	Start conveyor belts manually
	Stop	BOOL	0.3	FALSE	Stop conveyor belts
	Reset	BOOL	0.4	FALSE	Set control to the basic setting
	Count	COUNTER	2.0		Counter for the parts
	Quantity	WORD	4.0	W#16#200	Number of parts
	Tim	TIMER	6.0		Timer for the monitor
	Dura1	S5TIME	8.0	S5T#5S	Monitoring time for parts
	Dura2	S5TIME	10.0	S5T#10S	Monitoring time for gap
OUT	Load	BOOL	12.0	FALSE	Load new parts onto belt
	Remove	BOOL	12.1	FALSE	Remove parts from belt
IN_OUT					
STAT	Belt1	Conveyor_belt	14.0		Control for belt 1
	Belt2	Conveyor_belt	20.0		Control for belt 2
	Belt3	Conveyor_belt	26.0		Control for belt 3
	Belt4	Conveyor_belt	32.0		Control for belt 4
	Check	Parts_counter	38.0		Control for counting and monitoring
TEMP					

The subsequent calls of the function blocks for conveyor belt control contain only the block parameters for the individual signals for each conveyor belt and the connection to the block parameters of "Feed". The individual signals are the light barriers, the commands for the belt motor and the motor faults. (We make use here of the fact that when a function block is called, not all block parameters have to be initialized.) We program the connections between the individual belt controllers using assignments.

The FB "Parts_counter" is called as a local instance even if it has no closer connection with the signals of the conveyor belt controls. The instance data block for "Feed" stores the FB data.

The input parameters *Quantity, Dura1* and *Dura2 for* "Feed" need to be set only once. This can be done using default values (as in the example) or in the restart routine in OB 100 (for example, through direct assignment if these three parameters are treated as global data).

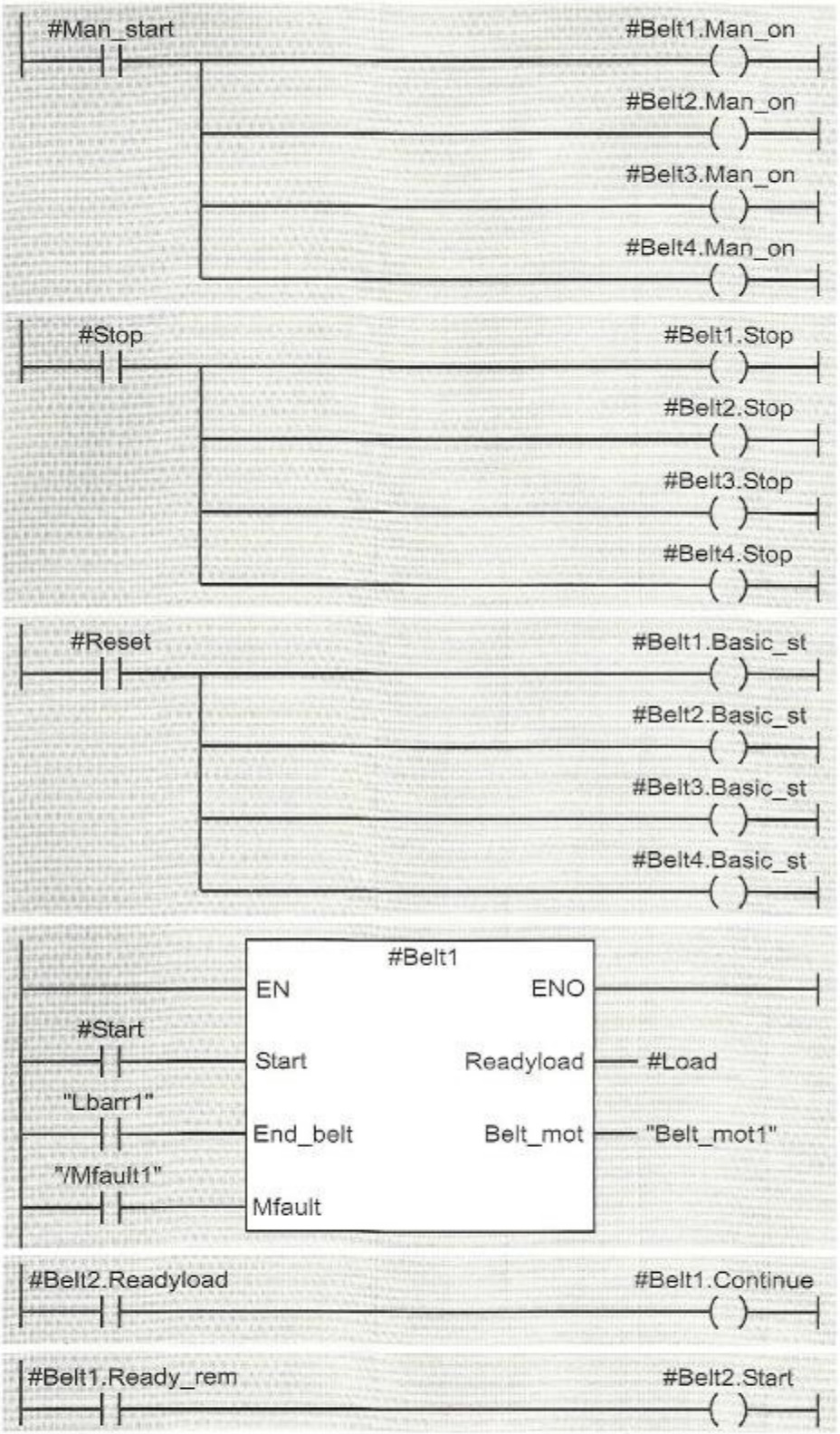
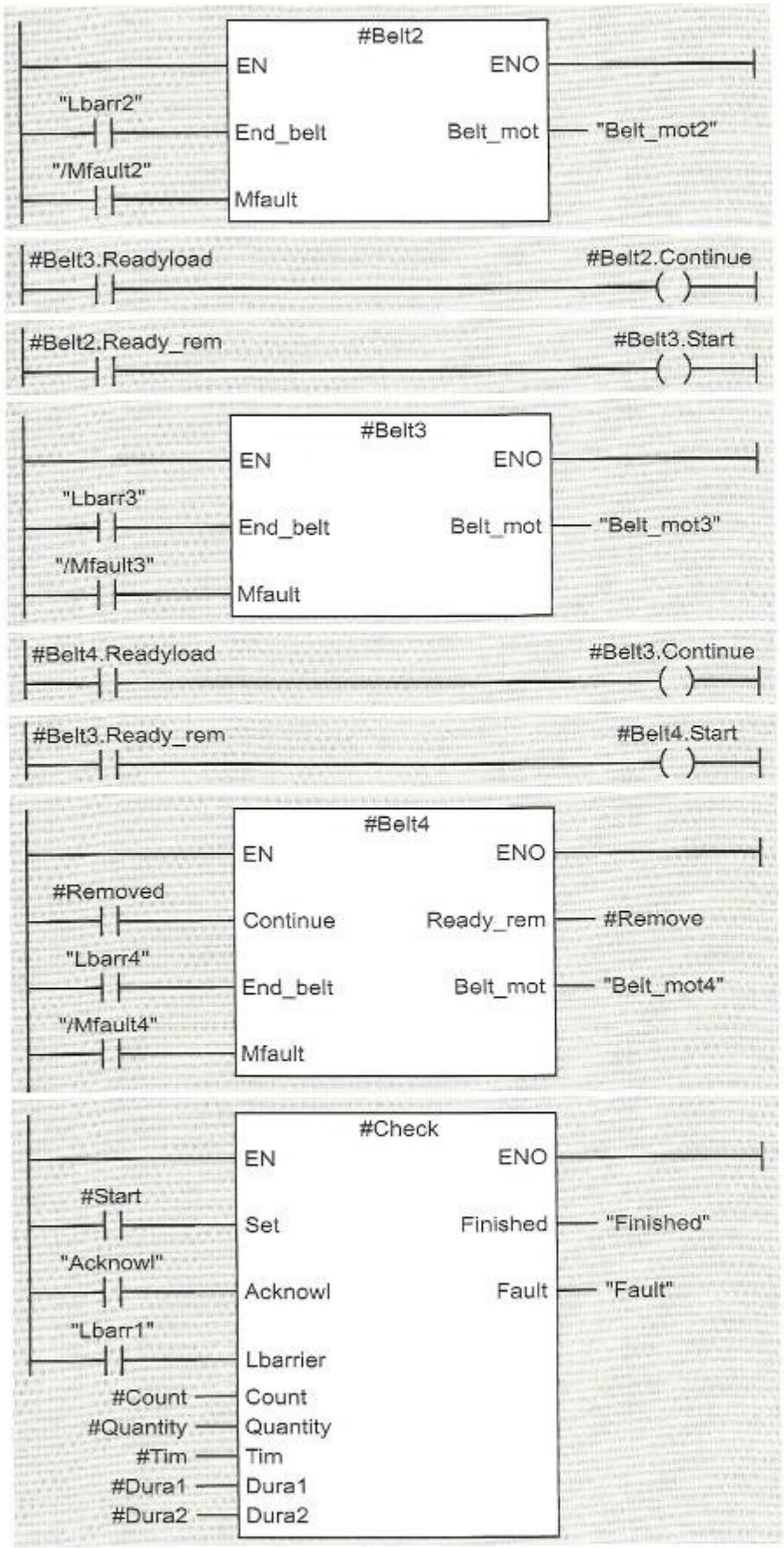


Figure 19.8 Program for Feed example (LAD)

Continued overleaf



Continued:

Figure19.8
Program for Feed Example (LAD)

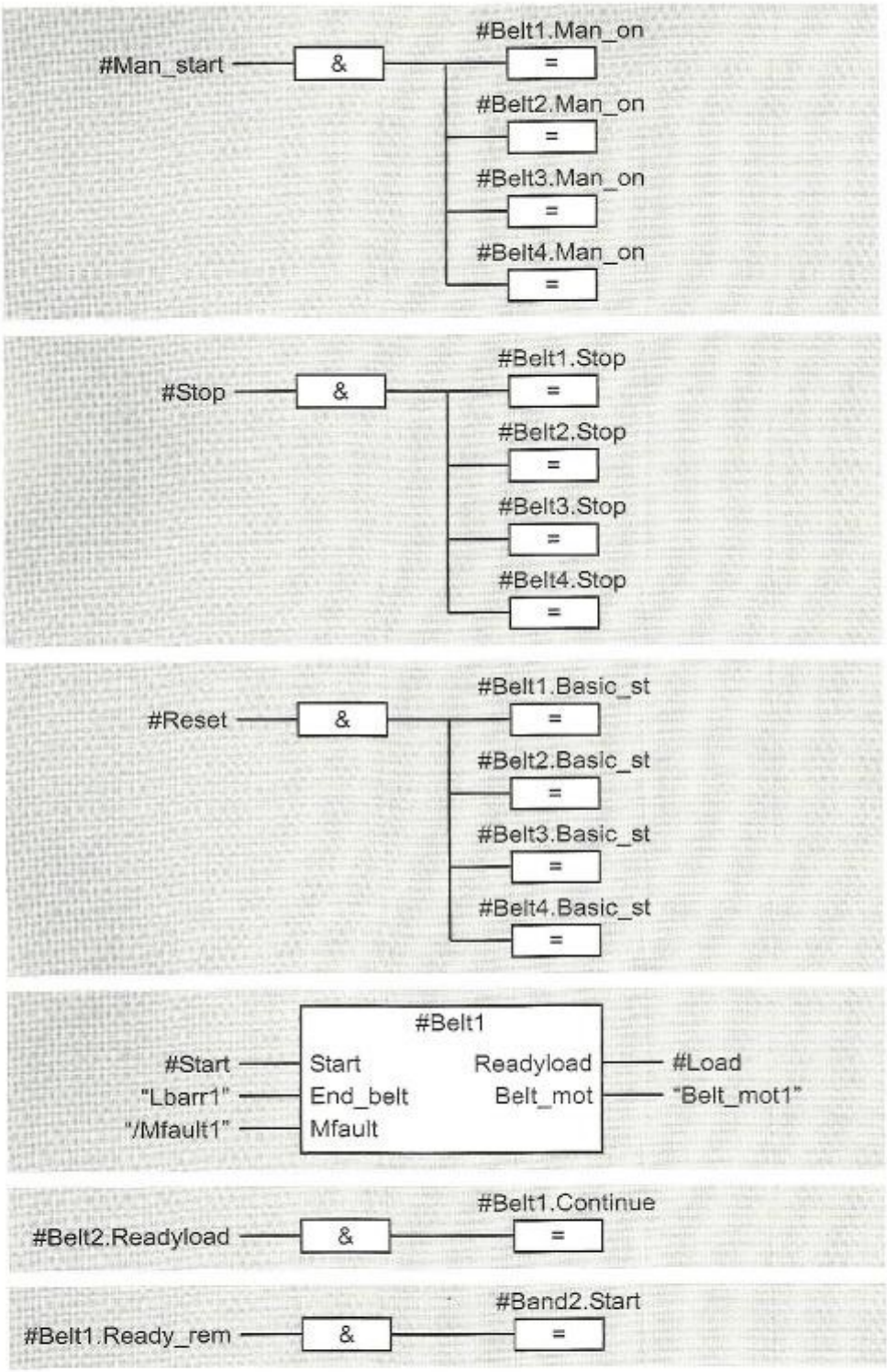
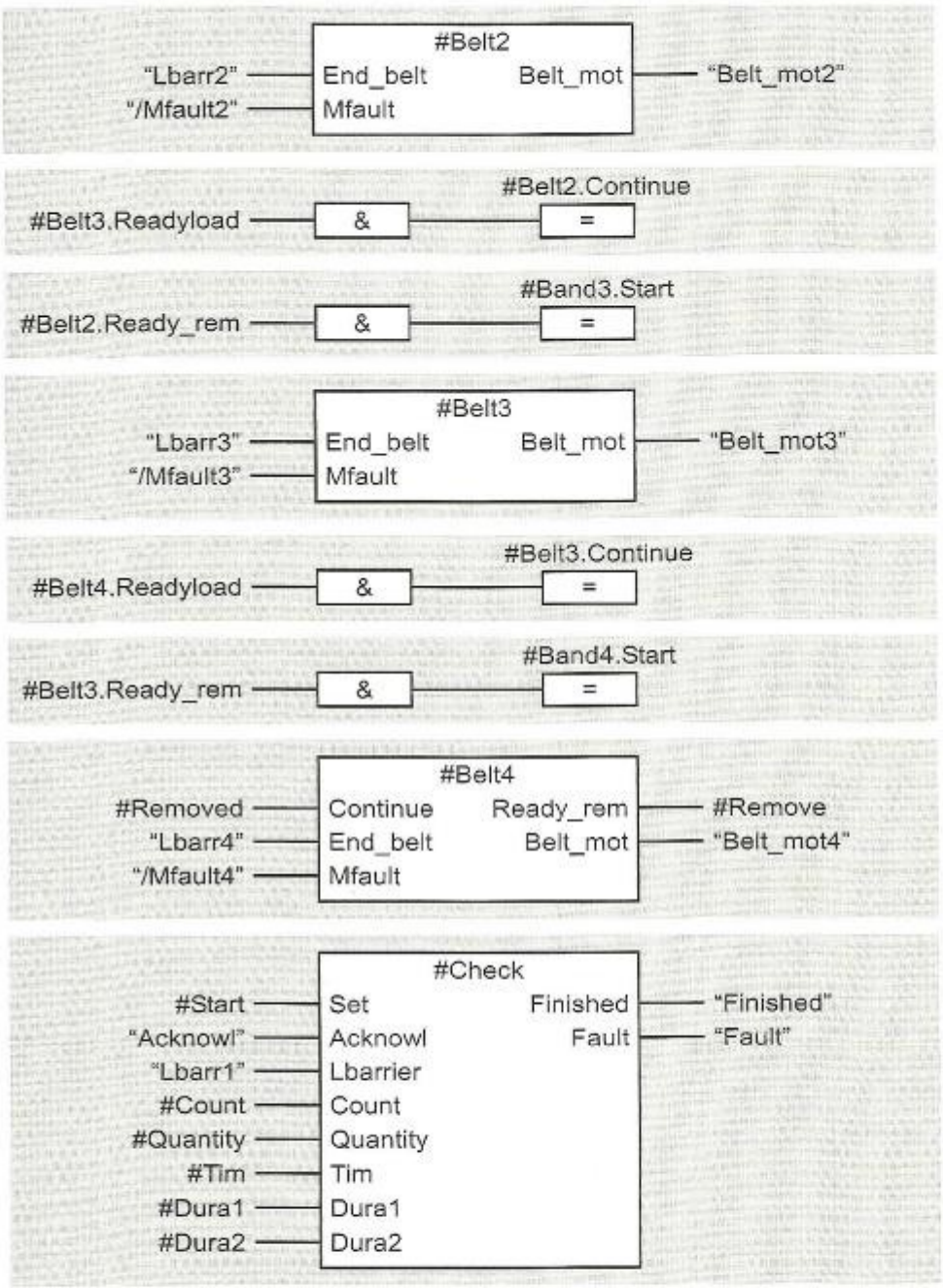


Figure 19.9 Program for Feed Example (FBD)

Continued overleaf



Continued:

Figure 19.9 Program for Feed Example (FBD)

Table 19.6 Symbol Table for the Conveyor Belt, Parts Counter and Feed Examples

Symbol	Address	Data Type	Comment
Conveyor_belt	FB 21	FB 21	Conveyor belt control
Belt_data1	DB 21	FB 21	Data for conveyor belt 1
Belt_data2	DB 22	FB 21	Data for conveyor belt 2
Parts_counter	FB 22	FB 22	Counter control and monitor
CounterDat	DB 29	FB 22	Data for parts counter
Feed	FB 20	FB 20	Feed with several belts
FeedDat	DB 20	FB 20	Data for Feed
Cycle	OB 1	OB 1	Main program, cyclic execution
Basic_st	I 0.0	BOOL	Set controller to the basic state
Man_on	I 0.1	BOOL	Switch on conveyor belt motors
/Stop	I 0.2	BOOL	Stop conveyor belt motors (zero active)
Start	I 0.3	BOOL	Start conveyor belt
Continue	I 0.4	BOOL	Acknowledgment that parts have been removed
Acknowl	I 0.6	BOOL	Acknowledge fault
Set	I 0.7	BOOL	Set counter, activate monitor
Lbarr1	I 1.0	BOOL	(Light barrier) "End of belt" sensor signal for conveyor belt 1
Lbarr2	I 1.1	BOOL	(Light barrier) "End of belt" sensor signal for conveyor belt 2
Lbarr3	I 1.2	BOOL	(Light barrier) "End of belt" sensor signal for conveyor belt 3
Lbarr4	I 1.3	BOOL	(Light barrier) "End of belt" sensor signal for conveyor belt 4
/Mfault1	I 2.0	BOOL	Motor protection switch conveyor belt 1 (zero-active)
/Mfault2	I 2.1	BOOL	Motor protection switch conveyor belt 2 (zero-active)
/Mfault3	I 2.2	BOOL	Motor protection switch conveyor belt 3 (zero-active)
/Mfault4	I 2.3	BOOL	Motor protection switch conveyor belt 4 (zero-active)
Readyload	Q 4.0	BOOL	Load new parts onto belt
Ready_rem	Q 4.1	BOOL	Remove parts from belt
Finished	Q 4.2	BOOL	Number of parts reached
Fault	Q 4.3	BOOL	Monitor activated
Belt_mot1	Q 5.0	BOOL	Switch on belt motor for conveyor belt 1
Belt_mot2	Q 5.1	BOOL	Switch on belt motor for conveyor belt 2
Belt_mot3	Q 5.2	BOOL	Switch on belt motor for conveyor belt 3
Belt_mot4	Q 5.3	BOOL	Switch on belt motor for conveyor belt 4
Quantity	MW 4	WORD	Number of parts
Dura1	MW 6	S5TIME	Monitoring time for light barrier covered
Dura2	MW 8	S5TIME	Monitoring time for light barrier not covered
Count	C 1	COUNTER	Counter for parts
Monitor	T 1	TIMER	Timer for monitor

Program Processing

This section of the book discusses the various methods of program processing.

The **main program** executes cyclically. After each program pass, the CPU returns to the beginning of the program and executes it again. This is the "standard" method of processing PLC programs.

Numerous system functions support the utilization of system services, such as controlling the real-time clock or communication via bus systems. In contrast to the static settings made when parameterizing the CPU, system functions can be used dynamically at program run time.

The main program can be temporarily suspended to allow **interrupt servicing**. The various types of interrupts (time-of-day interrupts, time-delay interrupts, watchdog interrupts, hardware interrupts, DPV1 interrupts, synchronous cycle interrupts, multiprocessor interrupt) are divided into priority classes whose processing priority you may yourself, to a large degree, determine. Interrupt servicing allows you to react quickly to signals from the controlled process or implement periodic control procedures independently of the processing time of the main program.

Before starting the main program, the CPU initiates a **start-up program** in which you can make specifications regarding program processing, define default values for variables, or parameterize modules.

Error handling is also part of program processing. STEP 7 distinguishes between synchronous errors, which occur during processing of a statement, and asynchronous errors, which can be detected independently of program processing. In both cases you can adapt the error routine to suit your needs.

20 Main program
Program structure; scan cycle control; response time; program functions; multicomputing operation; data exchange with system functions; start information

21 Interrupt handling
Time-of-day interrupts; time-delay interrupts; watchdog interrupts; hardware interrupts; DPV1 interrupts; multiprocessor interrupt; synchronous cycle interrupts; handling interrupt events

22 Start-up characteristics
Power-up, memory reset, retentivity; cold restart; warm restart; ascertain module address; parameterize modules

23 Error handling
Synchronous errors (programming errors, access errors); handling synchronous error events; asynchronous errors; system diagnostics

20 Main Program

The main program is the cyclically scanned user program; cyclic scanning is the "normal" way in which programs execute in programmable logic controllers. The large majority of control systems use only this form of program execution. If event-driven program scanning is used, it is in most cases only in addition to the main program.

The main program is invoked in organization block OB 1. It executes at the lowest priority level, and can be interrupted by all other types of program processing. The user program is executed in RUN mode, which is set using the mode selector on the CPU's front panel. With a toggle switch as mode selector, the position is RUN, with a key-operated switch, the positions are RUN and RUN-P. When in the RUN-P position, the CPU can be programmed via a programming device. In the RUN position, you can remove the key so that no one can change the operating mode without proper authorization; when the mode selector is at RUN, programs can only be read.

20.1 Program Organization

20.1.1 Program Structure

To analyze a complex automation task means to divide that task into smaller tasks or functions in accordance with the structure of the process to be controlled. You then define the individual tasks resulting from this dividing process by determining the functions and stipulating the interface signals to the process or to other tasks. This breakdown into individual tasks can be done in your program. In this way, the structure of your program corresponds to the division of the automation task.

A divided user program can be more easily configured, and can be programmed in sections (even by several people in the case of very large user programs). And finally, but not lacking in importance, dividing the program simplifies both debugging and service and maintenance.

The structuring of the user program depends on its size and its function. A distinction is made between three different "methods":

In a **linear program**, the entire main program is in organization block OB 1. Each current path is in a separate network. STEP 7 numbers the networks in sequence. When editing and debugging, you can reference every network directly by its number.

A **partitioned program** is basically a linear program which is divided into blocks. Reasons for dividing the program might be because it is too long for organization block OB 1 or because you want to make it more readable. The blocks are then called in sequence. You can also divide the program in another block the same way you would the program in organization block OB 1. This method allows you to call associated process-related functions for processing from within one and the same block. The advantage of this program structure is that, even though the program is linear, you can still debug and run it in sections (simply by omitting or adding block calls).

A **structured program** is used when the conceptual formulation is particularly extensive, when you want to reuse program functions, or when complex problems must be solved. Structuring means dividing the program into sections (blocks) which embody self-contained functions or serve a specific functional purpose and which exchange the fewest possible number of signals with other blocks. Assigning each program section a specific (process-related) function will produce easily readable blocks with simple interfaces to other blocks when programmed.

The LAD and FBD programming languages support structured programming through func-

tions with which you can create "blocks" (self-contained program sections). Chapter 3.2, "Blocks", discusses the different kinds of blocks and their uses. You will find a detailed description of the functions for calling and ending blocks in Chapter 18, "Block Functions". The blocks receive the signals and data to be processed via the call interface (the block parameters), and forward the results over this same interface. The options for passing parameters are described in detail in Chapter 19, "Block Parameters".

20.1.2 Program Organization

Program organization determines whether and in what order the CPU will process the blocks which you have generated. To organize your program, you program block calls in the desired sequence in the supraordinate blocks. You should chose the order in which the blocks are called so that it mirrors the process-related or function-related division of the controlled plant.

Nesting depth

The maximum depth applies for a priority class (for the program in an organization block), and is CPU-dependent. On the CPU 314, for example, the nesting depth is eight, that is, beginning with one organization block (nesting depth 1), you can add seven more blocks in the "horizontal" direction (this is called "nesting"). If more blocks are called, the CPU goes to STOP with a "Block overflow" error. Do not forget to include system function block (SFB) calls and system function (SFC) calls when calculating the nesting depth.

A data block call, which is actually only the opening or selecting of a data area, has no effect on the nesting depth of blocks, nor is the nesting depth affected by calling several blocks in succession (linear block calls).

Practice-related program organization

In organization block OB 1, you should call the blocks in the main program in such a way as to roughly organize your program. A program can be organized on either a process-related or function-related basis.

The following points of discussion can give only a rough, very general view with the intention of giving the beginner some ideas on program structuring and on translating his control task into reality. Advanced programmers normally have sufficient experience to organize a program to suit the special control task at hand.

A **process-related program structure** closely follows the structure of the plant to be controlled. The individual program sections correspond to the individual parts of the plant or of the process to be controlled. Subordinate to this rough structure are the scanning of the limit switches and operator panels and the control of the actuators and display devices (in different parts of the plant). Bit memory or global data

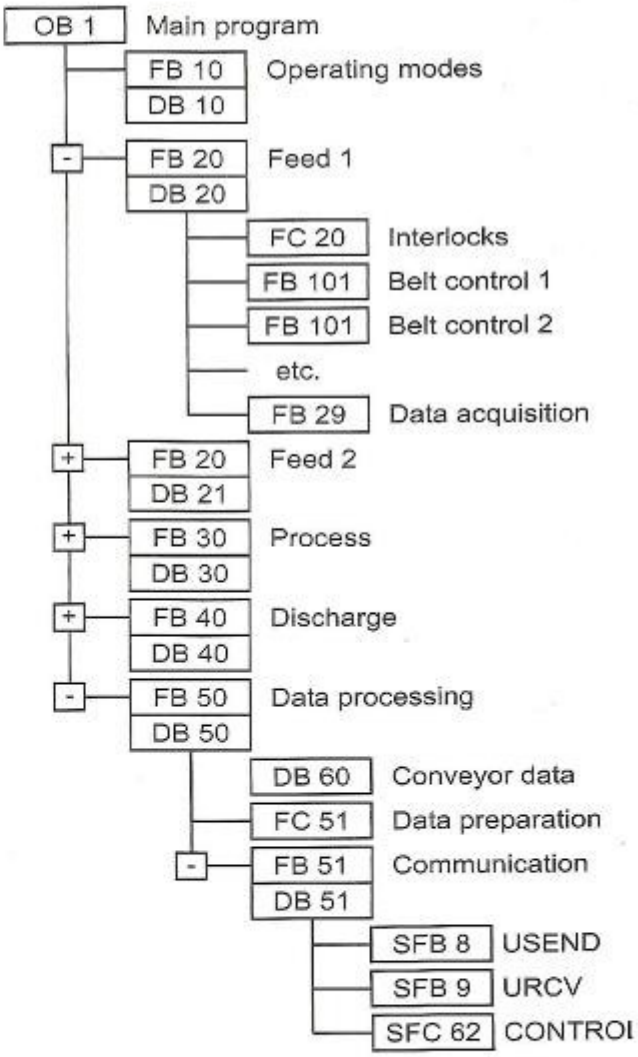


Figure 20.1 Example of Program Structuring

are used for signal interchange between different parts of the plant.

A **function-related program structure** is based on the control function to be executed. Initially, this method of program structuring does not take the controlled plant into account at all. The plant structure first becomes apparent in the subordinate blocks when the control function defined by the rough structure is divided further.

In practice, a hybrid of these two concepts is normally used. Figure 20.1 shows an example: A functional structure is mirrored in the operating mode program and in the data processing program which goes above and beyond the plant itself. Program sections Feeding Conveyor 1, Feeding Conveyor 2, Process and Discharging Conveyor are process-related.

The example also shows the use of different types of blocks. The main program is in OB 1; it is in this program that the blocks for the operating modes, the various pieces of plant equipment, and for data processing are called. These blocks are function blocks with an instance data block as data store. Feeding Conveyor 1 and Feeding Conveyor 2 are identically structured; FB 20, with DB 20 as instance data block for Feeding Conveyor 1 and with DB 21 as instance data block for Feeding Conveyor 2, is used for control.

In the conveyor control program, function FC 20 processes the interlocks; it scans inputs or memory bits and controls FB 20's local data. Function block FB 101 contains the control program for a conveyor belt, and is called once for each belt. The call is a local instance, so that its local data are in instance data block DB 20. The same applies for the data acquisition program in FB 29.

The data processing program in FB 50, which uses DB 50, processes the data acquired with FB 29 (and other blocks), which are located in global data block DB 60. Function FC 51 prepares these data for transfer. The transfer is controlled by FB 51 (with DB 51), in which system blocks SFB 8, SFB 9 and SFB 62 are called. Here, too, the SFBs save their instance data in "supraordinate" data block DB 51.

20.2 Scan Cycle Control

20.2.1 Process Image Updating

The process image is part of the CPU's internal system memory (Chapter 1.1.6, "CPU Memory Areas"). It begins at I/O address 0 and ends at an upper limit stipulated by the CPU. On appropriately equipped CPUs, you can define this limit yourself.

Normally, all digital modules lie in the process image address area, while all analog modules have addresses outside this area. If the CPU has free address allocation, you can use the configuration table to direct any module over the process image or address it outside the process image area.

The process image consists of the process-image input table (inputs I) and the process-image output table (outputs Q).

After CPU restart and prior to the first execution of OB 1, the operating system transfers the signal states of the process-image output table to the output modules and accepts the signal states of the input modules into the process-image input table. This is followed by execution of OB 1 where normally the inputs are combined with each other and the outputs are controlled. Following termination of OB 1, a new cycle begins with the updating of the process image (Figure 20.2).

If an error occurs during automatic updating of the process image, e.g. because a module is no longer accessible, organization block OB 85 "Program Execution Errors" is called. If OB 85 is not available, the CPU goes to STOP.

Subprocess images

With appropriately equipped CPUs, you can divide the process image into up to 30 subprocess images. You make this division during parameterization of the signal modules by defining the subprocess image via which the module is to be addressed when you assign addresses. You can separate the division according to process-image input table and process-image output table.

All modules that you do not assign to one of the subprocess images 1 to 30 are stored in subprocess image 0, which is also called the OB1 pro-

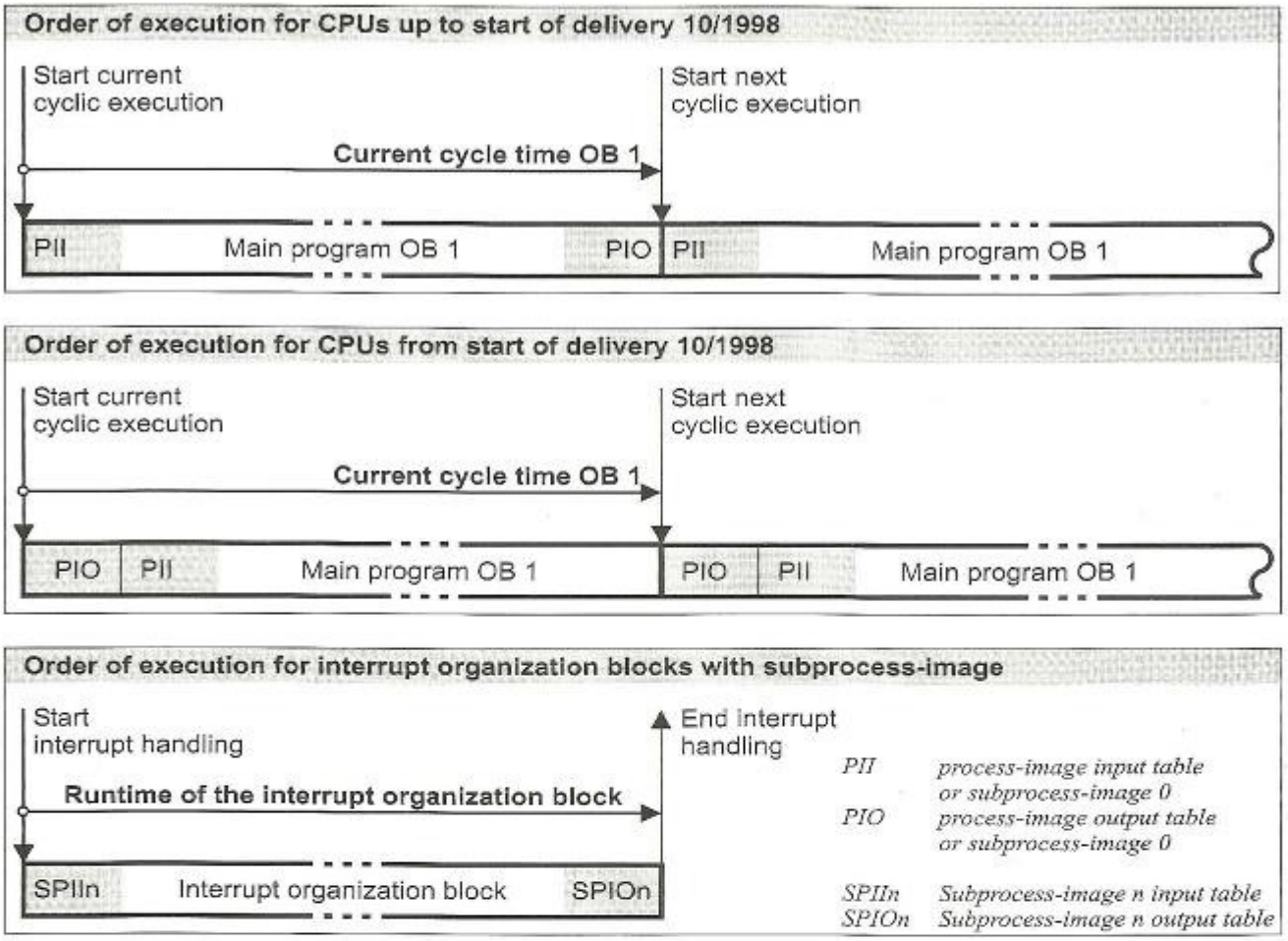


Figure 20.2 Updating the Process Image

cess image (OB1-PI). This subprocess image 0 is updated automatically by the operating system of the CPU as part of cyclic execution. You can also program the CPU such that the automatic updating is switched off.

With appropriately equipped CPUs, you can also assign the subprocess images to the interrupt organization blocks so that they are automatically updated when these OBs are called.

The system functions SFC 26 UPDAT_PI and SFC 27 UPDAT_PO are available for updating the subprocess images by the user program. In the organization blocks for synchronous cycle interrupt, you use the system functions SFC 126 SYNC_PI and SFC 127 SYNC_PO (Chapter 21.8.2, "Isochrone Updating Of Process Image").

SFC 26 UPDAT_PI
SFC 27 UPDAT_PO

The system function SFC 26 UPDAT_PI updates a subprocess image of the inputs, the system function SFC 27 UPDAT_PO updates a subprocess image of the outputs. Table 20.1 shows the parameters of these SFCs. You can also update subprocess image 0 with these SFCs.

You can carry out updating of individual subprocess images by calling these SFCs at any time and at any location. For example, you can define a subprocess image for a priority class (a program execution level) and you can then cause this subprocess image to be updated at the start and at the end of the relevant organization block when this priority class is processed.

Updating of a process image can be interrupted by calling a higher priority class. If an error

occurs during updating of a process image, e.g. because a module can no longer be accessed, this error is reported via the function value of the SFC.

20.2.2 Scan Cycle Monitoring Time

Program scanning in organization block OB 1 is monitored by the so-called "scan cycle monitor" or "scan cycle watchdog". The default value for the scan cycle monitoring time is 150 ms. You can change this value in the range from 1 ms to 6 s by parameterizing the CPU accordingly.

If the main program takes longer to scan than the specified scan cycle monitoring time, the CPU calls OB 80 ("Timing error"). If OB 80 has not been programmed, the CPU goes to STOP.

The scan cycle monitoring time includes the full scan time for OB 1. It also includes the scan times for higher priority classes which interrupt the main program (in the current cycle). Communication processes carried out by the operating system, such as GD communication or PG access to the CPU (block status!), also increase the runtime of the main program. The increase can be reduced in part by the way you parameterize the CPU ("Cyclic load from communication" on the "Cycle/Clock memory bits" tab).

Cycle statistics

If you have an online connection from a programming device to an operating CPU, select PLC → MODULE INFORMATION to call up a dialog box that contains several tabs. The "Cycle Time" tab shows the current cycle time as well as the shortest and longest cycle time. The parameterized minimum cycle duration and the scan cycle monitoring time are also displayed.

The cycle time for the last cycle and the minimum and maximum cycle time since the PLC was last started up can also be read in the temporary local data in the start information of OB 1.

SFC 43 RE_TRIGR
Restarting the scan cycle monitoring time

An SFC 43 RE_TRIGR system function call restarts the scan cycle monitoring time; the timer restarts with the new value set via CPU parameterization. SFC 43 has no parameters.

Operating system run times

The scan cycle time also includes the operating system run times. These are composed of the following:

- System control of cyclic scanning ("no-load cycle"), fixed value
- Updating of the process image; dependent on the number of bytes to be updated
- Updating of the timers; dependent on the number of timers to be updated
- Communications load

Communications functions for the CPU include the transfer of user program blocks or data exchange between CPU modules using system functions. The time the CPU is to use for these functions can be limited by parameterizing the CPU.

All values at operating system runtime are properties of the relevant CPU.

Table 20.1 Parameters for the SFCs for Process Image Updating

Parameter Name	SFC		Declaration	Data Type	Contents, Description
PART	26	27	INPUT	BYTE	Number of the subprocess image (0 to 15)
RET_VAL	26	27	RETURN	INT	Error information
FLADDR	26	27	OUTPUT	WORD	On an access error: the address of the first byte to cause the error

20.2.3 Minimum Scan Cycle Time, Background Scanning

With appropriately equipped CPUs, you may specify a minimum scan cycle time. If the main program (including interrupts) takes less time, the CPU waits until the specified minimum scan cycle time has elapsed before beginning the next cycle by recalling OB 1.

The default value for the minimum scan cycle time is 0 ms, that is to say, the function is disabled. You can set a minimum scan cycle time of from 1 ms to 6 s in "Cycle/Clock memory bits" tab when you parameterize the CPU.

Background scanning OB 90

In the interval between the actual end of the cycle and expiration of the minimum cycle time, the CPU executes organization block OB 90 "Background scanning" (Figure 20.3). OB 90 is executed "in slices". When the operating system calls OB 1, execution of OB 90 is interrupted; it is then resumed at the point of interruption when OB 1 has terminated. OB 90 can be interrupted after each statement, any system block called in OB 90, however, is first scanned in its entirety.

The length of a "slice" depends on the current scan cycle time of OB 1. The closer OB 1's scan time is to the minimum scan cycle time, the less time remains for executing OB 90. The program scan time is not monitored in OB 90.

OB 90 is scanned only in RUN mode. It can be interrupted by interrupt and error events, just like OB 1. The start information in the temporary local data (Byte 1) also tells which events cause OB 90 to execute from the beginning:

- B#16#91
  After a CPU restart;
- B#16#92
  After a block processed in OB 90 was deleted or replaced;
- B#16#93
  After (re)loading of OB 90 in RUN mode;
- B#16#95
  After the program in OB 90 was scanned and a new background cycle begins.

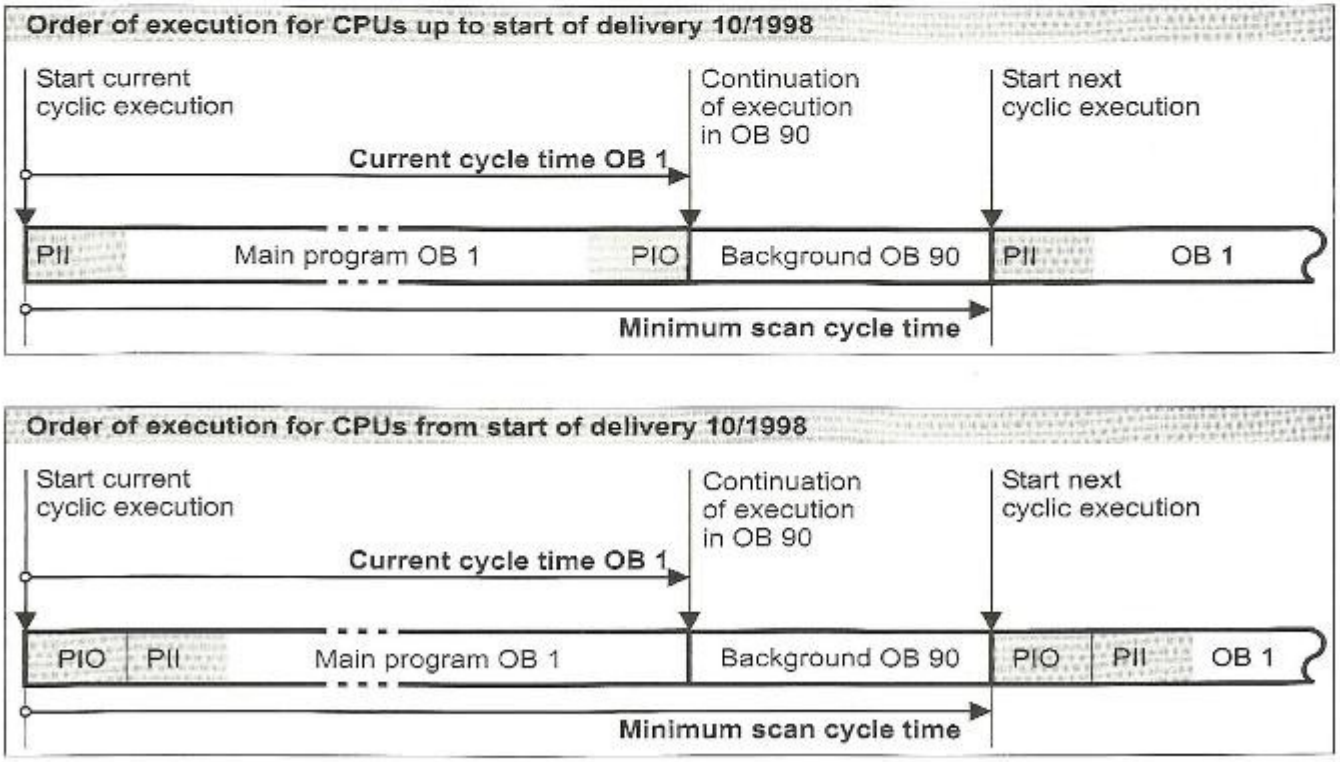


Figure 20.3 Minimum Cycle Duration and Background Scanning

20.2.4 Response Time

If the user program in OB 1 works with the signal states of the process images, this results in a response time which is dependent on the program execution time (scan cycle time). The response time lies between one and two scan cycles, as the following example explains.

When a limit switch is activated, for instance, it changes its signal state from "0" to "1". The programmable controller detects this change during the subsequent updating of the process image, and sets the inputs allocated to the limit switch to "1". The program evaluates this change by resetting an output, for example, in order to switch off the corresponding motor. The new signal state of the output that was reset is transferred at the end of the program scan; only then is the corresponding bit reset on the digital output module.

In a best-case situation, the process image is updated immediately following the change in the limit switch's signal (Figure 20.4). It would then take only one cycle for the relevant output to respond. In the worst-case situation, updating of the process image was just completed when the limit switch signal changed. It would then be necessary to wait approximately one cycle for the programmable controller to detect the signal change and set the input. After yet another cycle, the program can respond.

When so considered, the user program's execution time contains all procedures in one program cycle (including, for instance, the servicing of interrupts, the functions carried out by the operating system, such as updating timers, controlling the MPI interface and updating the process images).

The response time to a change in an input signal can thus be between one and two cycles. Added to the response time are the delays for the input modules, the switching times of contactors, and so on.

In some instances, you can reduce the response times by addressing the I/Os directly or calling program sections on an event-driven basis (process interrupt).

You can achieve uniform response times or equal time intervals for the process control if a program component is always processed at the same time interval, e.g. a cyclic interrupt program. Program processing isochronous with the processing cycle of a PROFIBUS DP master system also produces calculable response times (Chapter 21.8, "Synchronous Cycle Interrupts").

20.2.5 Start Information

The CPU's operating system forwards start information to organization block OB 1, as it does to every organization block, in the first 20 bytes of temporary local data. You can generate the declaration for the start information yourself or you can use information from *Standard Library* under *Organization Blocks*.

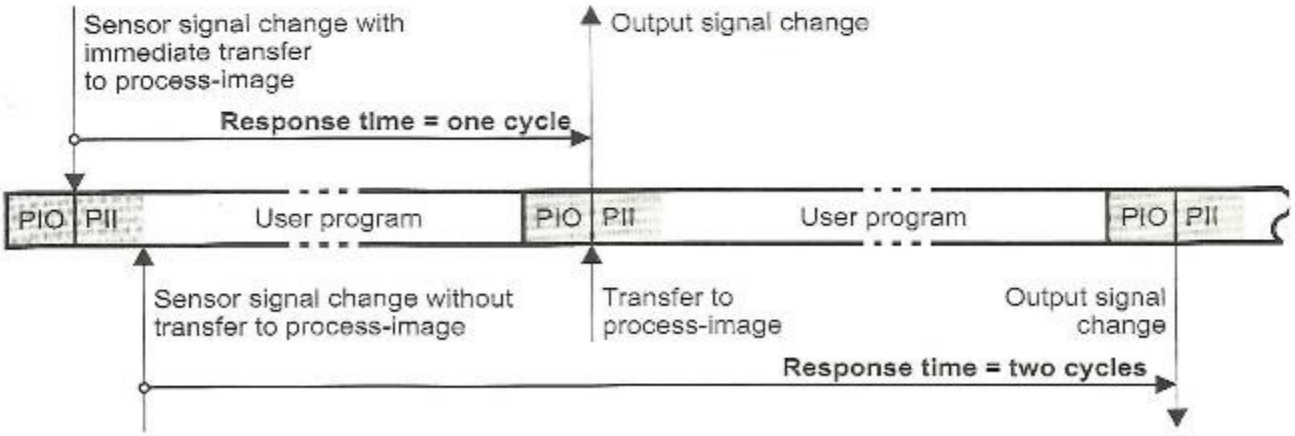


Figure 20.4 Response Times of Programmable Controllers

Table 20.2 Start Information for the OB 1

Byte	Variable Name	Data Type	Description	Contents
0	OB1_EV_CLASS	BYTE	Event class	B#16#11 = Call standard OB
1	OB1_SCAN_1	BYTE	Start information	B#16#01 = 1st cycle after complete restart B#16#02 = 1st cycle after warm restart B#16#03 = Every other cycle B#16#04 = 1st cycle after cold restart
2	OB1_PRIORITY	BYTE	Priority	B#16#01
3	OB1_OB_NUMBR	BYTE	OB Number	B#16#01
4	OB1_RESERVED_1	BYTE	Reserved	-
5	OB1_RESERVED_2	BYTE	Reserved	-
6..7	OB1_PREV_CYCLE	INT	Previous scan cycle time	in ms
8..9	OB1_MIN_CYCLE	INT	Minimum scan cycle time	in ms
10..11	OB1_MAX_CYCLE	INT	Maximum scan cycle time	in ms
12..19	OB1_DATE_TIME	DT	Event occurrence	Call time of the OB (cyclic)

Table 20.2 shows this start information for the OB1, the default symbolic designation, and the data types. You can change the designation at any time and choose names more acceptable to you. Even if you don't use the start information, you must reserve the first 20 bytes of temporary local data for this purpose (for instance in the form of a 20-byte array).

In SIMATIC S7, all event messages have a fixed structure which is specified by the event class. The start information for OB 1, for instance, reports event B#16#11 as a standard OB call. From the contents of the next byte you can tell whether the main program is in the first cycle after power-up and is therefore calling, for instance, initialization routines in the cyclic program.

The priority and OB number of the main program are fixed. With three INT values, the start information provides information on the cycle time of the last scan cycle and on the minimum and maximum cycle times since the last power-up. The last value, in DATE_AND_TIME format, indicates when the priority control program received the event for calling OB 1.

Note that direct reading of the start information for an organization block is possible only in that organization block because that information consists of temporary local data. If you require the start information in blocks which lie on deeper levels, call system function SFC RD_SINFO at the relevant location in the program.

SFC 6 RD_SINFO
Reading out start information

System function SFC 6 RD_SINFO makes the start information on the current organization block (that is, the OB at the top of the call tree) and on the start-up OB last executed available to you even at a deeper call level (Table 20.3).

Output parameter TOP_SI contains the first 12 bytes of start information on the current OB, output parameter START_UP_SI the first 12 bytes of start information on the last start-up OB executed. There is no time stamp in either case.

SFC 6 RD_SINFO can not only be called at any location in the main program but in every priority class, even in an error organization block or in the start-up routine. If the SFC is called in an interrupt organization block, for example, TOP_SI contains the start information of the interrupt OB. In the case of a call at restart, TOP_SI and START_UP_SI have the same contents.

Table 20.3 Parameters for SFC 6 RD_SINFO

SFC	Parameter Name	Declaration	Data Type	Contents, Description
6	RET_VAL	RETURN	INT	Error information
	TOP_SI	OUTPUT	STRUCT	Start information for the current OB (with the same structure as START_UP_SI)
	START_UP_SI	OUTPUT	STRUCT	Start information for the last OB started:
	.EV_CLASS		BYTE	Event ID and event class
	.EV_NUM		BYTE	Event number
	.PRIORITY		BYTE	Execution priority (number of the execution level)
	.NUM		BYTE	OB number
	.TYP2_3		BYTE	ID of supplementary information 2_3
	.TYP1		BYTE	ID of supplementary information 1
	.ZI1		WORD	Supplementary information 1
	.ZI2_3		DWORD	Supplementary information 2_3

20.3 Program Functions

In addition to parameterizing the CPU with the Hardware Configuration, you can also select a number of program functions dynamically at runtime via the integrated system functions.

20.3.1 Time of day

Each SIMATIC CPU has a clock which you can set and scan using STEP 7 or system functions. The time of day is represented in the user program in the format DATE_AND_TIME, consisting of the date, time and day of week.

Newer CPUs with firmware version 3 and higher also have the time status. You can then additionally set a difference from a time zone as well as summer/winter time identification.

The following system functions can be used to access the CPU clock:

- ▷ SFC 0 SET_CLK
  Set date and time
- ▷ SFC 1 READ_CLK
  Read date and time
- ▷ SFC 48 SNC_RTCB
  Synchronize CPU clocks
- ▷ SFC 100 SET_CLKS
  Set time of day, date and time status

You will find a list of system function parameters in Table 20.4.

Setting and reading the time of day

Calling the SFC 0 SET_CLK or the SFC 100 SET_CLKS using MODE = B#16#01 or B#16#03 set the clock to the value defined by the PDT parameter. SFC 0 SET_CLK sets the winter time for CPUs with summer/winter time identification. With the SFC 100 SET_CLKS, you can use the SUMMER parameter to define whether the time is the winter time (with "0") or the summer time (with "1").

The current time is read by SFC 1 READ_CLK and output in the CDT parameter. The time has the format DATE_AND_TIME when setting and reading, i.e. therefore contains the date and time.

Module time, local time

The time present on the CPU is the module time. This is decisive for all timed processes which control the CPU, such as run-time meters, starting of time-of-day interrupts or entering of time stamps in the diagnostics buffer and OB start information. You can set and read the module time using the system functions for the CPU clock.

Correspondingly designed CPUs additionally save a "time status". This contains a correction value which results in the local time when added to the module time. The correction value is set in intervals of 30 minutes, and can also be negative (CORR parameter of SFC 100 SET_CLKS).

The local time can be used to display time zones.

Table 20.4 SFC Parameters for the CPU Clock

SFC	Parameter Name	Declaration	Data Type	Contents, Description
0	PDT	INPUT	DT	Date and time (new)
	RET_VAL	RETURN	INT	Error information
1	RET_VAL	RETURN	INT	Error information
	CDT	OUTPUT	DT	Date and time (current)
48	RET_VAL	RETURN	INT	Error information
100	MODE	INPUT	BYTE	Operating mode B#16#01: Only set time B#16#02: Only set time status B#16#03: Set time and time status
	PDT	INPUT	DT	Defined time
	CORR	INPUT	INT	Difference from basic time in 0.5-hour intervals from –24 to +26
	SUMMER	INPUT	BOOL	Summer/winter time identification ("1" = summer time)
	ANN_1	INPUT	BOOL	Announcement of time switchover: a "1" indicates that a switch is made from summer time to winter time or vice versa the next time that the hour changes
	RET_VAL	RETURN	INT	Error information

Time status

The time status is set when parameterizing the CPU with STEP 7 or with the SFC 100 SET_CLKS. The time of day and the time status can be read using the SFC 51 RDSYSST via the system status list (SSL_ID = W#16#0132 with INDEX W#16#0008). The *status* variable includes:

- The correction value (bits 2 to 6) in the 30-minute interval
- The sign of the correction value (bit 7)
- The summer/winter time identification (bit 14)
- The announcement hour (bit 15)

The *summer/winter time identification* shows whether the local time calculated from the module time and the correction value is the summer time (with "1") or the winter time (with "0").

If the *announcement hour* bit has the status "1", the switchover from summer time to winter time is carried out at the next change in hour.

Using the time status information, a local time can be generated from the module time in order to control timed processes in the user program.

Loadable standard blocks help you to handle the summer/winter time switching of the local time in the user program, in particular the starting of time-of-day interrupts depending on the local time (see further below under, "Loadable time-of-day blocks").

Time synchronization

In an automation network with several SIMATIC stations exchanging data with one another on subnets, it is possible to synchronize the clocks of all CPUs. You parameterize the clock of one CPU as the "Master clock", and set the interval at which the synchronization is to be carried out. You parameterize the blocks to be synchronized as "Slave clocks".

The synchronization can be carried out within an S7 station over the communications bus (backplane bus) or between stations over the MPI bus. This is carried out at the parameterized interval automatically when the master clock is set for the first time. If you set a master clock with the SFC 0 SET_CLK or SFC 100 SET_CLKS, all other clocks in the subnetwork are automatically synchronized to this value.

By calling the SFC 48 SNC_RTCB in the master clock, you can synchronize all slave blocks independent of the automatic interval.

If the master clock does not have a time-of-day status, the slave clocks are synchronized with the winter time. The correction factor is zero, the local time then corresponds to the module time.

If the master clock works with a time-of-day status, the complete time status is transmitted in addition to the time. Therefore the same local time (the same time zone) exists on all CPUs in the time network.

Set time using STEP 7

When setting the CPU parameters, you can set the synchronization mode (master, slave or none) and the synchronization interval in the attributes window on the "Diagnostics/clock" tab). The correction value set here is used to set the clock accuracy.

You can set the time and the time status using STEP 7 if the programming device is connected online to a CPU. Select PLC → DIAGNOSTICS/ CUSTOMIZE → SET TIME OF DAY. In the enhanced dialog, you can set the local time as a difference from the module time, and define the summer/winter time. The time status is shown in the "Status" box.

Loadable time-of-day blocks

The program *Miscellaneous Blocks* in the *Standard Library* contains loadable blocks for handling summer/winter time switching and the local time in the user program.

- ▷ FC 60 LOC_TIME
  Determine local time
- ▷ FC 61 BT_LT
  Convert module time into local time
- ▷ FC 62 LT_BT
  Convert local time into module time
- ▷ FC 63 S_LTINT
  Set time-of-day interrupt to local time
- ▷ FB 60 SET_SW
  Switch over summer/winter time
- ▷ FB 61 SET_SW_S
  Switch over summer/winter time with time status
- ▷ UDT 60 WS_RULES
  Rules for switching over summer/winter time (e.g. time for switching over)

20.3.2 Read System Clock

A CPU's system clock starts running on power-up. The system clock keeps running as long as the CPU is executing the restart routine or is in RUN mode. When the CPU goes to STOP or HOLD, the current system time is "frozen".

If you initiate a hot restart on an S7-400 CPU, the system clock starts running again using the saved value as its starting time. Cold restart or warm restart reset the system time.

The system time has data format TIME, whereby it can assume only positive values:

TIME#0ms to
TIME#24d20h31m23s647ms.

In the event of an overflow, the clock starts again at 0. Newer CPUs update the system clock every millisecond, older S7-300-CPUs every 10 milliseconds.

SFC 64 TIME_TCK
Read system time

You can read the current system time with system function SFC 64 TIME_TCK. The RET_VAL parameter contains the system time in the TIME data format.

You can use the system clock, for example, to read out the current CPU runtime or, by computing the difference, to calculate the time between two SFC 64 calls. The difference between two values in TIME format is computed using DINT subtraction.

20.3.3 Run-Time Meter

A run-time meter in a CPU counts the hours. You can use the run-time meter for such tasks as determining the CPU runtime or ascertaining the runtime of devices connected to that CPU.

The value on the run-time meter is also retained following a cold restart, failure of the backup voltage, and following a memory reset.

Table 20.5 Parameters of the SFCs for the Run-Time Meter

SFC	Parameter	Declaration	Data Type	Contents, Description
2	NR	INPUT	BYTE	Number of the run-time meter (B#16#00 to B#16#07)
	PV	INPUT	INT	New value for the run-time meter
	RET_VAL	RETURN	INT	Error information
3	NR	INPUT	BYTE	Number of the run-time meter (B#16#00 to B#16#07)
	S	INPUT	BOOL	Start (with "1") or stop (with "0") run-time meter
	RET_VAL	RETURN	INT	Error information
4	NR	INPUT	BYTE	Number of the run-time meter (B#16#00 to B#16#07)
	RET_VAL	RETURN	INT	Error information
	CQ	OUTPUT	BOOL	Run-time meter running ("1") or stopped ("0")
	CV	OUTPUT	INT	Current value of the run-time meter
101	NR	INPUT	BYTE	Number of the run-time meter (B#16#00 to B#16#0F)
	MODE	INPUT	BYTE	Order code (see text)
	PV	INPUT	DINT	New value for the run-time meter
	RET_VAL	RETURN	INT	Error information
	CQ	OUTPUT	BOOL	Run-time meter running ("1") or stopped ("0")
	CV	OUTPUT	DINT	Current value of the run-time meter

The range of values and the number of run-time meters per CPU depend on the CPU. The range is 16 bits (2^{15}– 1 hours) or 32 bits (2^{31}– 1 hours). When the CPU is at STOP or HOLD, the run-time meter also stops running; when the CPU is restarted, the run-time meter must be restarted.

When a run-time meter reaches the maximum time duration, it stops and reports an overflow. A run-time meter can be set to a new value or reset to zero only via an SFC call.

The following system functions are available to control a run-time meter:

- ▷ SFC 2 SET_RTM
  Set 16-bit run-time meter
- ▷ SFC 3 CTRL_RTM
  Start or stop 16-bit run-time meter
- ▷ SFC 4 READ_RTM
  Read 16-bit run-time meter
- ▷ SFC 101 RTM
  Adjust 32-bit run-time meter

Table 20.5 shows the parameter for these system functions.

The NR parameter stands for the number of the run-time meter, and has the data type BYTE. It can be initialized using a constant or a variable (as can all input parameters of elementary data type). The PV parameter (data type INT) is used to set the run-time meter to an initial value. SFC 3's-S-parameter starts (with signal state "1") or stops (with signal state "0") the selected run-time meter. CQ indicates whether the run-time meter was running (signal state "1") or stopped (signal state "0") when scanned. The CV parameter records the hours in INT format.

By assigning the MODE parameter of SFC 101, you can control a 32-bit run-time meter as follows:

B#16#00	Read current meter value
B#16#01	Start at last meter value
B#16#02	Stop meter
B#16#04	Set to value specified in PV
B#16#05	Set and start at value specified in PV
B#16#06	Set and stop at value specified in PV

You can also use the SFCs for a 16-bit run-time meter to control a 32-bit run-time meter. The

latter then responds as with a 16-bit range of values.

20.3.4 Compressing CPU Memory

Multiple deletion and reloading of blocks, which often occur during online block modification, can result in gaps in the CPU's work memory and in the RAM load memory which decrease the amount of usable space in memory. When you call the "Compress" function, you start a CPU program which fills these gaps by pushing the blocks together. You can initiate the "Compress" function via a programming device connected to the CPU or by calling system function **SFC 25 COMPRESS**. The parameters for SFC 25 are listed in Table 20.6.

The compression procedure is distributed over several program cycles. The SFC returns BUSY = "1" to indicate that it is still in progress, and DONE = "1" to indicate that it has completed the compression operation. The SFC cannot compress when an externally initiated compression is in progress, when the "Delete Block" function is active, or when PG functions are accessing the block to be shifted (for instance the Block Status function).

Note that blocks of a particular CPU-specific maximum length cannot be compressed, so that gaps would still remain in CPU memory. Only the Compress function initiated via the PG while the CPU is at STOP closes all gaps.

20.3.5 Waiting and Stopping

The system function **SFC 47 WAIT** halts the program scan for a specified period of time.

SFC 47 WAIT has input parameter WT of data type INT in which you can specify the waiting time in microseconds (μs).

The maximum waiting time is 32767 µs; the minimum waiting time corresponds to the execution time of the system function, which is CPU-specific.

SFC 47 can be interrupted by higher-priority events. On an S7-300, this increases the waiting time by the scan time of the higher-priority interrupt routine.

The system function **SFC 46 STP** terminates the program scan, and the CPU goes to STOP. SFC 46 STP has no parameters.

20.3.6 Multiprocessing Mode

The S7-400 enables multiprocessing. As many as four appropriately designed CPUs can be operated in one rack on the same P bus and K bus.

An S7-400 station is automatically in multiprocessor mode if you arrange more than one CPU in the central rack in the Hardware Configuration. The slots are arbitrary; the CPUs are distinguished by a number assigned automatically in ascending order when the CPUs are plugged in. You can also assign this number yourself on the "Multicomputing" tab.

The configuration data for all the CPUs must be loaded into the PLC, even when you make changes to only one CPU.

After assigning parameters to the CPUs, you must assign each module in the station to a CPU. This is done by parameterizing the module in the "Addresses" tab under "CPU Allocation" (Figure 20.5).

At the same time that you assign the module's address area, you also assign the module's interrupts to this CPU. With VIEW → FILTER → CPU No. X-MODULES, you can emphasize the modules assigned to a CPU in the configuration tables.

Table 20.6 Parameters for SFC 25 COMPRESS

SFC	Parameter	Declaration	Data Type	Contents, Description
25	RET_VAL	RETURN	INT	Error information
	BUSY	OUTPUT	BOOL	Compression still in progress (with "1")
	DONE	OUTPUT	BOOL	Compression completed (with "1")

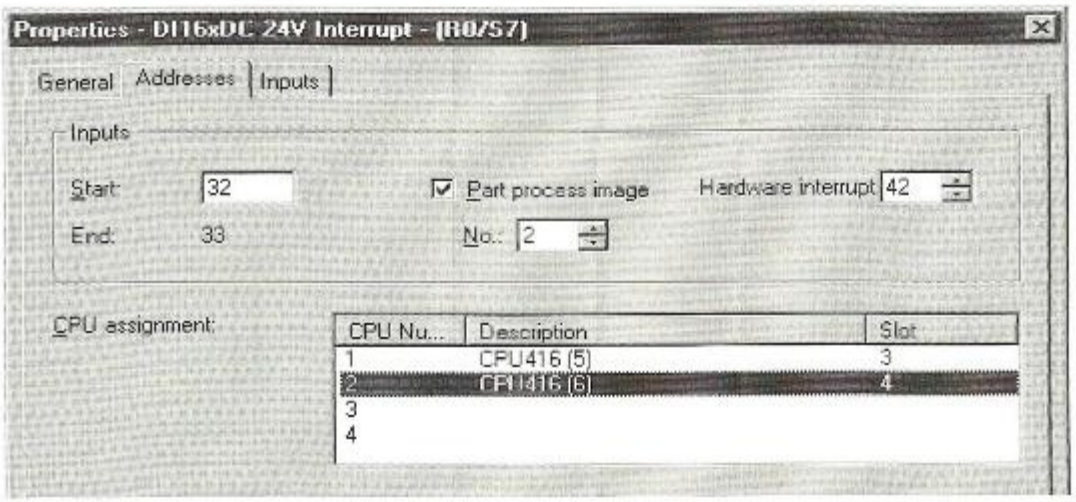


Figure 20.5 Module Assignments in Multiprocessor Mode

The CPUs in a multiprocessing network all have the same operating mode. This means

- They must all be parameterized with the same restart mode;
- They all go to RUN simultaneously;
- They all go to HOLD when you debug in single-step mode in one of the CPUs;
- They all go to STOP as soon as one of the CPUs goes to STOP.

When one rack in the station fails, organization block OB 86 is called in each CPU.

The user programs in these CPUs execute independently of one another; they are not synchronized.

An SFC 35 MP_ALM call starts organization block OB 60 "Multiprocessor interrupt" in all CPUs simultaneously (see Chapter 21.7, "Multiprocessor Interrupt").

20.3.7 Determining the OB Program Runtime

The system function **SFC 78 OB_RT** determines the runtime of individual organization blocks over different time periods. This enables you to determine the time load (utilization) of the user program.

The operating system of a CPU designed for this purpose logs the runtimes of the individual organization blocks and makes them available for reading via the SFC 78 OB_RT. The accuracy of the time acquisition is CPU-dependent and the times are specified in microseconds. If there is no value pending for a requested time, –1 (DW#16#FFFF FFFF) is returned.

Principle of time measurement

In the operating system of the CPU, a timer runs with a relative time in microseconds from 0 to $2^{31}-1$. At the transition from STOP to RUN, the timer is started, runs to the upper limit, and then starts again from zero.

The OB start event, the beginning and end of OB execution, and the interruptions caused by higher-priority OBs are captured in the operating system. The data of the last completed OB execution that was current at the time of calling the SFC 78 is stored.

SFC call outside the OB to be measured

When applying the SFC, a distinction is made between a call in the program of the requested OB and one outside the requested OB. Example: The SFC 78 is called in the OB 1 and is assigned a value of 30 in the parameter OB_NR. The last captured times for OB 30 are then read. Specification of the synchronous error OB with the numbers 121 and 122 is not permissible because these belong to the priority class of the error-causing OBs and thus to their program.

Figure 20.6 shows some examples of calling SFC 78 outside the OB to be measured. The ini-

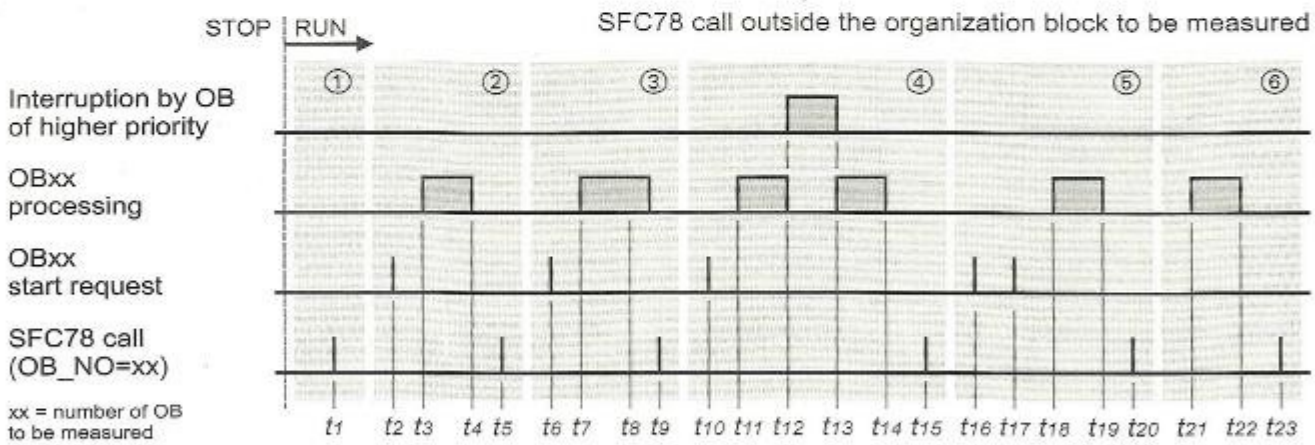
Table 20.7 Parameters of the SFC 78

SFC	Parameter	Declaration	Data Type	Contents, Description
78	OB_NR	INPUT	INT	Number of the OB whose timers are to be scanned
	RET_VAL	RETURN	INT	Error information or number of the OB when OB_NR=0
	PRIO	OUTPUT	INT	Priority class of the scanned OB
	LAST_RT	OUTPUT	DINT	Runtime of the last completed execution
	LAST_ET	OUTPUT	DINT	Period between OB request and end of processing for the last completed execution
	CUR_T	OUTPUT	DINT	Period of OB request (relative time)
	CUR_RT	OUTPUT	DINT	Previous runtime of the OB execution
	CUR_ET	OUTPUT	DINT	Period between OB request and scanning by SFC 78 in the requested OB
	NEXT_ET	OUTPUT	DINT	Period since the next OB request and scanning by the SFC 78

tial values after a STOP-RUN transition are –1 (example ①).

LAST_RT indicates the runtime in microseconds of the last completed OB execution (examples ② to ⑥). The "net" runtimes are output. Interrupt times caused by OBs with higher priority classes are not included in LAST_RT (④).

LAST_ET indicates the time period in microseconds between the start request and the end of processing for the last completed execution of



Values in the SFC78 parameters

Name	①	②	③	④	⑤	⑥
LAST_RT	–1	t4 - t3	t8 – t7	(t14 – t13) + (t12 – t11)	t19 – t18	t22 – t21
LAST_ET	–1	t4 - t2	t8 – t6	t14 – t10	t19 – t16	t22 – t17
CUR_T	–1	0	0	0	0	0
CUR_RT	–1	0	0	0	0	0
CUR_ET	–1	0	0	0	0	0
NEXT_ET	–1	–1	–1	–1	–1	–1

Figure 20.6 Calling SFC 78 outside the organization block to be measured

the OB to be measured (examples ② to ⑥). Interrupt times caused by higher priority classes are included in LAST_ET (④).

CUR_T indicates the relative time in microseconds (status of the counter in the operating system) of the start request of the OB. After initialization CUR_T (①) contains –1. On completion of OB execution, CUR_T is set to zero . Since SFC 78 is called outside the OB in these examples it consequently outputs zero at this parameter.

CUR_RT indicates the effective execution time of the OB until calling of SFC 78 in microseconds. After initialization CUR_RT (①) contains –1. After completion of OB execution, the value in CUR_RT is transferred to LAST_RT and CUR_RT is set to zero. Since the SFC call takes place outside the OB in these examples, the value is always zero.

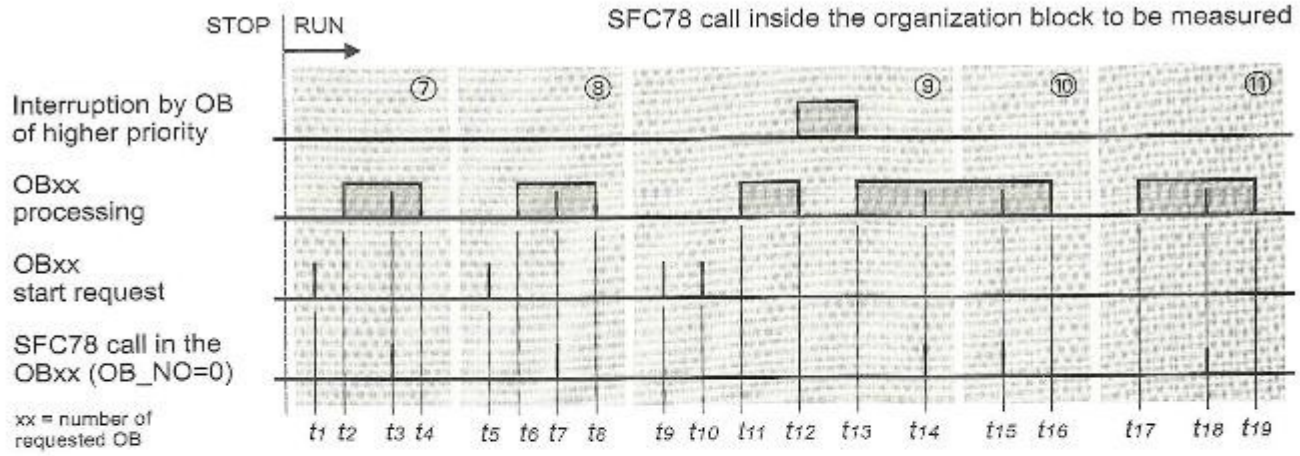
CUR_ET indicates the time period from the OB start request to calling of SFC 78 in microseconds. After completion of OB execution, the value in CUR_ET is transferred to LAST_ET and CUR_ET is set to zero. Since the SFC call takes place outside the OB in these examples, the value is always zero.

NEXT_RT indicates the time from the next OB start request to calling of the SFC in microseconds if further, unprocessed start requests are pending. In the case of the currently supplied CPUs, NEXT_RT is not calculated and the value is always –1.

SFC call in the program of the OB to be measured

SFC 78 can also be called in the program of the OB to be measured. The parameter OB_NR is then initialized with zero and the parameter RET_VAL returns the current OB number – in the case of error-free execution. The times for the OB in which SFC 78 is called are read. When calling in one of the synchronous error OBs with the numbers 121 and 122, this is the data of the OB causing the error.

Figure 20.7 shows some examples of calling SFC 78 in the program of the OB to be measured. This can be directly in the OB program



Values in the SFC78 parameters

Name	⑦	⑧	⑨	⑩	⑪
LAST_RT	–1	t4 – t2	t8 – t6	–1	(t16 – t13) + (t12 – t11)
LAST_ET	–1	t4 – t1	t8 – t5	–1	t16 – t9
CUR_T	t1	t5	t9	t9	t10
CUR_RT	t3 – t2	t7 – t6	(t14 – t13) + (t12 – t11)	(t15 – t13) + (t12 – t11)	t18 – t17
CUR_ET	t3 – t1	t7 – t5	t14 – t9	t15 – t9	t18 – t10
NEXT_ET	–1	–1	–1	–1	–1

Figure 20.7 Calling SFC 78 within the organization block to be measured

Table 20.8 Parameters of SFC 109 PROTECT

SFC	Parameter	Declaration	Data type	Contents, Description
109	MODE	INPUT	WORD	Job ID: W#16#0000 Protection level 1 W#16#0001 Protection level 2
	RET_VAL	RETURN	INT	Error information

or in one of the blocks called there. The initial values after a STOP-RUN transition are –1.

LAST_RT indicates the runtime in microseconds of the last completed OB execution (examples ⑧, ⑨ and ⑪). If the SFC is called again in the OB to be measured, –1 is output (example ⑩). The "net" runtimes are output; interrupt times caused by higher priority classes are not included in LAST_RT (example ⑨).

LAST_ET indicates the time period in microseconds between the request and the end of processing for the last completed execution of the OB to be measured (examples ⑧ and ⑨). This also applies for the first call of SFC 78 in the OB to be measured (⑨). If the SFC is called again in the OB to be measured, –1 is output (example ⑩). LAST_ET also contains the interrupt times caused by higher priority classes (⑨).

CUR_T indicates the relative time in microseconds (status of the counter in the operating system) of the start request of the OBs, when – as in the following examples – the SFC 78 is called within the OB. On completion of OB execution, CUR_T is set to zero.

CUR_RT indicates the effective execution time of the OB in microseconds until calling of SFC 78. After completion of OB execution, the value in CUR_RT is transferred to LAST_RT and CUR_RT is set to zero. Interrupt times caused by higher priority classes are not included in CUR_RT (⑨ and ⑩).

CUR_ET indicates the time period from the OB start request to calling of SFC 78 in microseconds. After completion of OB execution, the value in CUR_ET is transferred to LAST_ET and CUR_ET is set to zero. CUR_ET also contains the runtimes of the higher-priority OBs that interrupt the OB currently to be measured.

NEXT_RT indicates the time from the next OB start request to calling of the SFC in microseconds if further, unprocessed start requests are pending. In the case of the currently supplied CPUs, NEXT_RT is not determined and the value is always –1.

20.3.8 Changing program protection

The user program in a CPU can be protected against access in three protection levels (see Chapter 2.6.2, "Protecting the User Program"). Program-driven toggling between protection levels 1 and 2 is possible with the system function **SFC 109 PROTECT**. You can find the parameters of this system function in Table 20.8.

Calling SFC 109 PROTECT is only effective if you have set Protection level 1 with the hardware configuration. It remains ineffective if Protection level 2 or 3 is set, or if a password has been entered in Protection level 1 in conjunction with the option "Can be revoked by password".

The protection level set with SFC 109 PROTECT remains unchanged if:

- ▷ the CPU goes to STOP due to a (program) error, an SFC 46 STP call, or operator intervention,
- ▷ the CPU is battery-backed and the mains supply is restored, or
- ▷ a restart is carried out (S7-400).

In all other cases, Protection level 1 is set in the case of an operating mode transition. Even if you switch the mode selector to STOP, Protection level 1 is (re)set.

You can determine the current protection level online in the SIMATIC Manager with selected CPU and PLC → DIAGNOSTIC/SETTING → OPERATING MODE. In the program, you can scan the protection level with the SFC 51 RDSYSST via the system status list W#16#0232 with the Index W#16#0004.

In the case of a CPU with key lock switch as mode selector, you can remove the key in the RUN position and permit read access only by the programming device during operation. In the case of a toggle switch as mode selector, this is not possible. In this case, you can call SFC 109 PROTECT in the restart organization blocks to activate Protection level 2, that is, write protection (read access only) when switching on the CPU.

With SFC 109 PROTECT, you can change the protection level during operation without operating the mode selector. Thus, for example, with SFC 109 PROTECT you can (re)set the protection level to 1 with MODE = W#16#0000 depending on the signal state of a binary variable, in order, for example, to reload program sections. Then you re-activate write protection with MODE = W#16#0001.

20.4 Communication via Distributed I/O

Distributed I/O is understood to be modules connected over PROFIBUS DP or PROFINET IO.

With PROFIBUS DP, the DP master communicates with the DP slaves assigned to it over the PROFIBUS subnetwork. With PROFINET IO, it is the IO controller which exchanges data with the IO devices assigned to it over the Industrial Ethernet subnetwork.

The data transmission is carried out "automatically", you do not need to be involved. You configure and address the distributed I/O using the Hardware Configuration tool in a manner similar to with the central modules. From the user program, you can address the inputs and outputs in the stations of the distributed I/O just like the inputs and outputs of the central modules.

20.4.1 Addressing PROFIBUS DP

The distributed modules (stations, DP slaves) with PROFIBUS DP are assigned to a DP master in a manner similar to how the central modules are assigned to a CPU and controlled by it. The DP master with all "its" DP slaves is referred to as the DP master system. Several DP master systems can be present in an S7 station.

Newer DP masters can handle two operating modes: DPV1 and S7-compatible. "S7-compatible" corresponds to the previous mode. With this, you can operate all DP standard slaves according to EN 50170, and additionally the "DP S7 slaves" from Siemens which could already send interrupts to the DP master. In DPV1 mode, you can additionally used DP slaves which exhibit the new properties according to IEC 61131, such as increased diagnostics and parameterization capabilities resulting from the acyclic transmission of data records or the use of new types of interrupt. New system functions for transmission of data records, as well as new interrupt organization blocks, exist for these new "DPV1 slaves".

Like the central modules, the DP slaves occupy addresses in the I/O area of the CPU ("logical address area"). The DP master is as it were "transparent" to the addresses of the DP slaves; the CPU "sees" the addresses of the DP slaves, meaning that these must not overlap with those of the central modules, not even with those of DP slaves in other DP master systems assigned to the CPU.

Every DP slave has three addresses in addition to the node address: a geographical address, a module starting address and at least one diagnostics address (Figure 20.8).

Node address

Every node on the PROFBIBUS subnetwork has a unique address, the node address (station number) in that subnetwork that distinguishes it from the other nodes on the subnetwork. The station (the DP master or a DP slave) is accessed on PROFIBUS with this node address.

Please note that there must be a gap of at least 1 between the addresses of the active bus nodes (e.g. in the case of DP master and nodes in cross traffic). STEP 7 takes this into account when assigning node addresses automatically.

Geographical address

The geographical address identifies a module slot. With central modules, the geographical

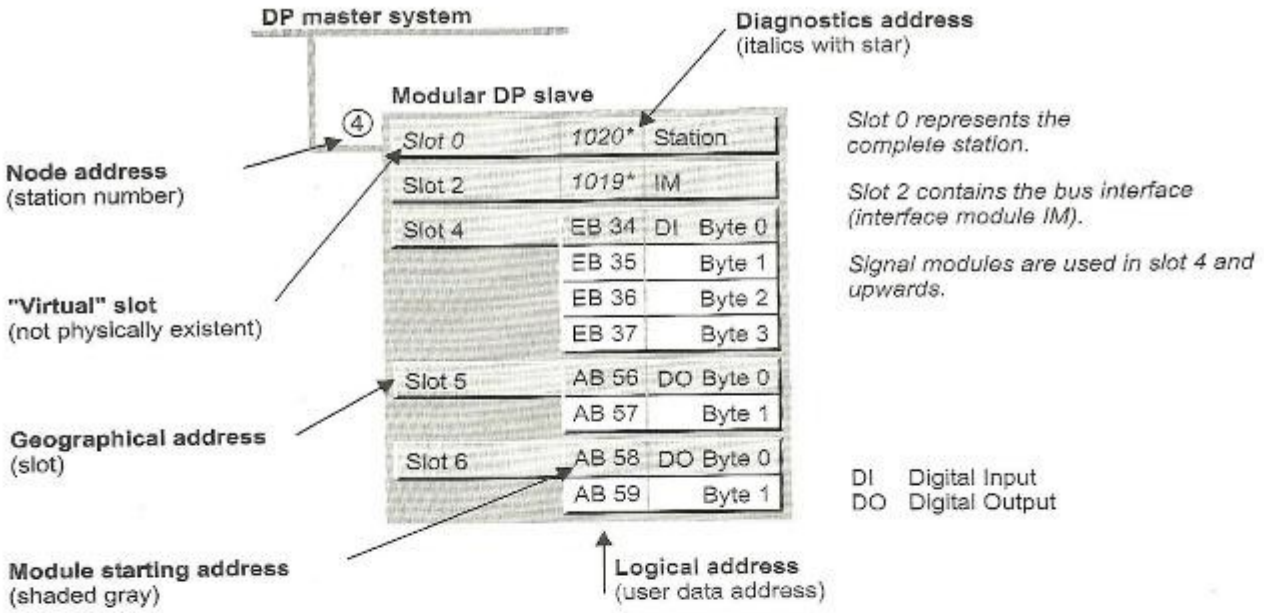


Figure 20.8 Addresses in a DP Master System

address contains the number of the rack and that of the slot. With PROFIBUS DP, the geographical address contains the number of the DP master system, the station number and the slot number.

Slot numbering of a DP slave depends on its type. If it is integrated using a GSD file, the entries in the GSD file determine the slot at which the I/O modules start. In the case of DP standard slaves, the slots for I/O modules begin at 1. Slot numbering of an S7 slave is based on the slots of an S7-300 station. Slots 1 (power supply) and 3 (expansion interface module) remain empty. Slot 2 (CPU) corresponds to the interface module (header module) of the modular DP slaves. The signal modules (SM) are positioned starting at slot 4. In addition, there is the "virtual" slot 0 (not physically present); this represents the complete station.

It is similar with intelligent DP slaves. In this case, the transfer memory is the interface to the DP master. Configuration of the transfer memory – which you carry out with the Hardware Configuration tool – generates areas which correspond to modules or slots. These slots do not really exist, and one therefore speaks of "virtual" slots.

Virtual slot 0 represents the DP station, virtual slot 2 the bus interface, in this case the slave CPU as the "header module" of the DP slave. From virtual slot 4 onwards, the user data areas are present in the transfer memory; they correspond to the signal modules. The virtual slots in the transfer memory are "seen" by both the master and slave CPUs.

The definition of virtual slots makes it possible to directly assign diagnostics and interrupt events of the interface module or the station (see further below "Diagnostics address"). The system functions SFC 5 GADR_LGC and SFC 49 LGC_GADR are available for conversion from the geographical address to the logical addresses and vice versa, and the system functions SFC 70 GEO_LOG and SFC 71 LOG_GEO are also suitable for PROFINET IO.

Note that DP slaves which are incorporated into the hardware configuration through a GSD file according to EN 50170 of version 3 or later (DPV1) can save the user data starting at slot 1.

Logical address, module starting address

You use the logical address to access the user data of a station. Each byte of the user data is unambiguously identified by a logical address. The logical address corresponds to the absolute address; a symbol (name) can be assigned to it so that it is easier to read (symbolic addressing).

The smallest logical address of a module or station is the module starting address (see also Chapter 1.4, "Module Addresses").

Diagnostics address

Modules and stations which can deliver diagnostics data and do not have a user data address themselves are identified by the diagnostics address. The diagnostics address occupies one byte of I/O input in the logical address volume. In the default setting, STEP 7 assigns the diagnostics address starting with the highest address in the I/O area of the CPU. You can change the diagnostics address. The address overview in the Hardware Configuration tool identifies the diagnostics address by a star.

Signal modules or user data areas in the transfer memory of intelligent DP slaves have logical addresses, also for scanning diagnostics data. The complete station delivers its diagnostics data via one diagnostics address which is assigned to virtual slot 0. With modular and intelligent DP slaves, the bus interface can deliver its diagnostics data via the diagnostics address of slot 2.

Figure 20.9 shows an example of the diagnostics addresses in a DP master system. A compact DP slave possesses one diagnostics address for the complete station, a modular DP slave possesses one diagnostics address for the station and one for the interface module. With intelligent DP slaves, there is also a diagnostics address for the DP interface.

The diagnostics addresses are assigned in descending order starting with the highest I/O address. For example, the DP interface of the CPU 317-2PN/DP is assigned the diagnostics address 8191, the PN interface the address 8190 (not shown in Figure 20.9), and the virtual slots 0 and 2 in the transfer memory the addresses 8189 and 8188. It is similar with the master CPU: the diagnostics addresses start at 16383 for the DP interface and continue with 16382 for the MPI/DP interface (not shown), 16381 and 16380 for the virtual slots 0 and 2 of the first intelligent DP slave, 16379 and 16378 for the second DP slave etc. The diagnostics addresses for the DP slaves are assigned by the Hardware Configuration tool in the sequence of coupling to the DP master system.

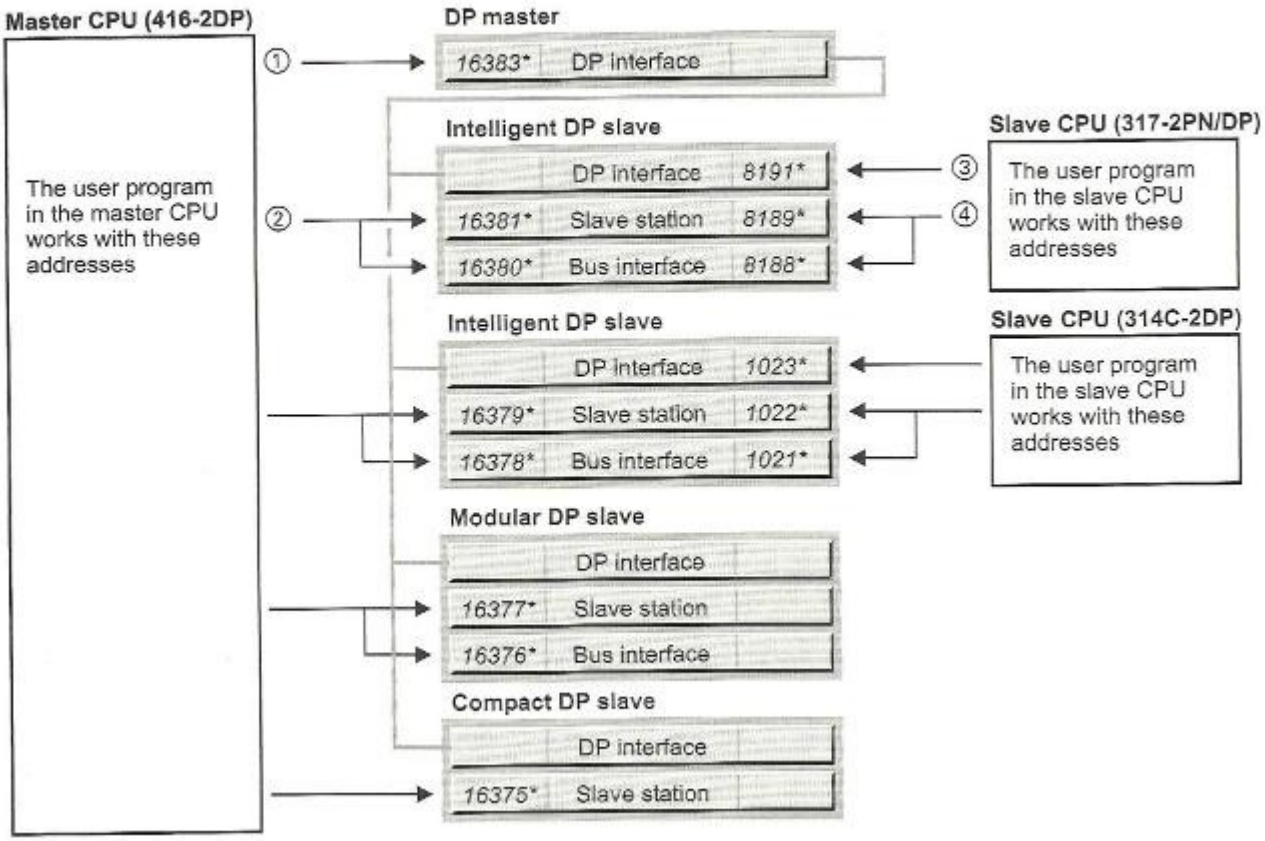
In the user program, the diagnostics data are scanned using system blocks. The system function SFC 13 DPNRM_DG is available for this with conventional DP standard slaves. The SFC 59 RD_REC is used with DP S7 slaves in order to read the data record DS1 with the diagnostics data. DPV1 slaves are able to provide more comprehensive diagnostics data which can be read with the system function block SFB 52 RDREC. The modules are addressed by means of the logical module starting address of the user data or by means of the diagnostics addresses.

Transfer memory on intelligent DP slaves

In the case of compact and modular DP slaves, the addresses of the inputs and outputs are located together with the addresses for the central modules in the address volume of the master CPU. In the case of intelligent slaves, the master CPU has no direct access to the input/output modules of the DP slave. Every intelligent DP slave therefore has a transfer memory whose size depends on the CPU used. The transfer memory can be divided into several areas of different length and data consistency. The individual areas then respond like modules whose lowest address is the module starting address. From the viewpoint of the master CPU, the intelligent DP slave then appears as a compact or modular DP slave, depending on the division (Figure 20.10).

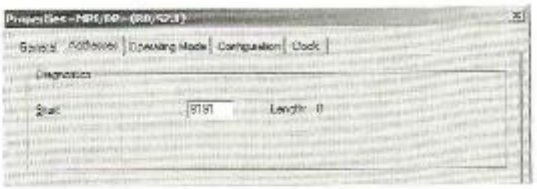
When configuring the slave, you can configure the individual areas of the transfer memory as inputs or outputs with the "module starting address" and the area length. Exception: if the CP 342-5DP provides the DP interface for the intelligent slave, the division of its transfer memory is only configured when coupling to the DP master system. The addresses of the transfer memory must not overlap with those of the central modules in the intelligent DP slave. If the addresses are present in the process image, the areas can be handled by the user program like inputs and outputs, otherwise like peripheral inputs and outputs. If the slave CPU possesses partial process images, you can assign a partial process image to each area.

When coupling to the DP master system, you supplement the configuration at the master end by the "module starting addresses" from the

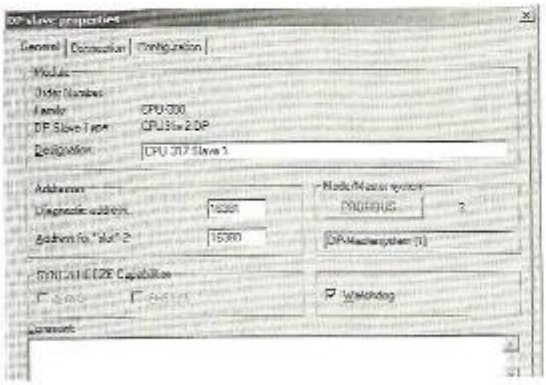


① Each bus interface has a diagnostics address which is set in the object properties of the interface in the "Addresses" tab (double click "DP" line in the master CPU).

③ Each bus interface has a diagnostics address which is set in the object properties of the interface in the "Addresses" tab (double click the "DP" line in the slave CPU).



② The station and ist bus interface have a diagnostics address which is set in the object properties of the station in the "General" tab (double click the DP slave station in the hardware configuration of the DP master).



④ The station and ist bus interface have a diagnostics address which is set in the object properties of the station in the "Mode" tab (double click the "DP" line in the slave CPU).

Figure 20.9 Diagnostics Addresses in a DP Master System

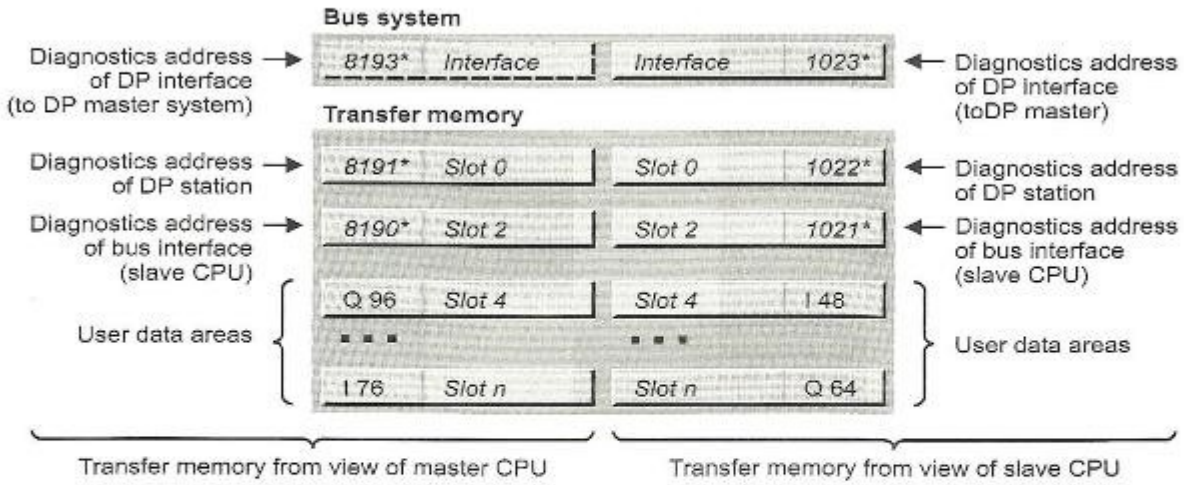


Figure 20.10 Transfer Memory on Intelligent DP Slaves

viewpoint of the master CPU and by the transmission direction. You assign inputs on the slave side to outputs on the master side and vice versa. If the addresses are within the process image, the areas can be handled by the user program like inputs and outputs, otherwise like peripheral inputs and outputs. If the master CPU possesses partial process images, you can also assign a partial process image to each area. From the viewpoint of the master CPU, the addresses of the transfer memory must not overlap with addresses of other modules in the (centralized) S7 station, neither with the addresses of the central modules nor with the addresses in other DP master systems assigned to the master CPU.

You also define the diagnostics addresses from the viewpoint of the intelligent DP slaves during configuration of these. You determine the diagnostics addresses from the viewpoint of the master CPU during coupling of the intelligent DP slave to the DP master system.

Data consistency

Data consistency means that a block of user data is handled all together. DP slaves can exhibit consistent data areas, either as a result of the configuration information in the GSD file or – as with the transfer memory of an intelligent slave – through configuration by the Hardware Configuration tool. In the object properties of the DP slave, one can set the data consistency to "Unit" (byte or word) or "Total" (configured area) on the "Configuration" tab. These data are taken into consideration by the operating system, e.g. during updating of the (partial) process image (see Chapter 1.5.3, "Consistent User Data").

As the user, you must ensure the consistency of the transmitted data in the program. If the data area comprises one, two or four bytes, transmit the data as bytes, words or doublewords using the load and transfer instructions or the MOVE box. With an area length of 3 or more than 4 bytes, use the system functions SFC 14 DPRD_DAT and SFC 15 DP_WR_DAT in order to transmit the data between the user memory and the transfer memory. If the addresses of the transfer memory are in the process image, use the address area for inputs I or outputs Q on the RECORD parameter of the SFCs, otherwise the address area for peripheral inputs PI or peripheral outputs PQ. The system functions are used in both the user program of the slave CPU and in the user program of the master CPU in order to access the data.

The configured data areas which are transmitted with the system functions are present completely in the process image (with newer CPUs) so that you can also access them using load and transfer instructions or the MOVE box. Note that you may destroy the data consistency in the process.

Starting from the user program of the slave CPU, you can trigger a process interrupt for an address area of the transfer memory using the

SFC 7 DP_PRAL, and a diagnostics or process interrupt in the master CPU using the SFB 75 SALRM.

20.4.2 Configuring PROFIBUS DP

General procedure

You configure the distributed I/O on the PROFIBUS DP in essentially the same way as the centralized modules. Instead of arranging modules in a mounting rack, you assign DP stations (PROFIBUS nodes) to a DP master system. The following order is recommended for the necessary actions:

1) Create a new project or open an existing one with the SIMATIC Manager.
2) Create a PROFIBUS subnetwork in the project with the SIMATIC Manager and, if required, set the bus profile.
3) Use the SIMATIC Manager to create the master station in the project that is to accommodate the DP master, e.g. an S7-400 station.

   If your system contains intelligent DP slaves, you also create the relevant slave stations at this point, e.g. S7-300 stations.

   You start the Hardware Configuration by opening the master station.
4) With the Hardware Configuration, you place a DP master in the master station. This can be, for example, a CPU with integral DP interface. You assign the previously created PROFIBUS subnetwork to the DP interface and you then have a DP master system. You also define the DP mode in the "Mode" tab: DPV1 or S7-compatible. You can also configure the remaining modules later. Save and compile the station.
5) If you have created an S7 station for an intelligent DP slave, you open this in the Hardware Configuration and you "plug in" the module with the desired DP interface, e.g. an S7-300 CPU with integral DP interface or an ET200pro basic module IM 154-8/CPU. If you set the DP interface as "DP slave", assign the previously created PROFIBUS subnetwork to the DP interface and configure the user data interface from the viewpoint of the DP slave (transfer memory). You can also configure the remaining modules later. Save and compile the station.

   Proceed in the same way for the remaining stations intended for intelligent DP slaves.
6) Open the master station with the DP master system and use the mouse to drag the PROFIBUS nodes (compact and modular DP slaves) from the hardware catalog to the DP master system. Assign node addresses and, if necessary, set the module starting address and the diagnostics address.
7) If you have created intelligent DP slaves, drag the relevant icon (in the hardware catalog under "PROFIBUS DP" and "Already configured stations") with the mouse to the DP master system.

   Open the icon and assign the already configured DP slave ("Connect"), assign a node address and configure the user data interface from the viewpoint of the DP master (or from the viewpoint of the central master CPU). Proceed in the same way with every intelligent DP slave.
8) Save and compile all stations. The DP master system is now configured. You can now supplement the configuration with centralized modules or with further DP slaves.

You can also represent the DP master system configured in this way graphically with the Network Configuration tool. Open Network Configuration by, for example, double-clicking on a subnetwork. Select VIEW → DP SLAVES/IO DEVICES to display the slaves. You can also create a DP master system (or more precisely, assign the nodes to a PROFIBUS subnetwork) with the Network Configuration tool. You parameterize the stations after opening them with the Hardware Configuration. Here too, you must first set up an intelligent DP slave before you can integrate it into a DP master system.

Configuring the DP master

You must have created a project and an S7 station with the SIMATIC Manager. You open the S7 station and create a mounting rack (see Chapter 2.3, "Configuring Stations"). Now drag the DP master module from the Hardware

Catalog to the configuration table of the mounting rack. You may already have selected a CPU with DP connection. In the line below, the DP master is displayed with a connection to a DP master system in the station window (broken black-and-white bar).

When placing the DP master module, you select in a window the PROFIBUS subnetwork to which the DP master system is to be assigned and the node address to be assigned to the DP master. You can also create a new PROFIBUS subnetwork in this window.

In the "Mode" tab you also define the DP mode with which the DP master is to work. This mode applies to the complete DP master system.

If there is no DP master system available (it may be that it is obscured behind an object or it is outside the visible area), create one by selecting the DP master in the configuration window and then selecting INSERT → MASTER SYSTEM. You can change the node address and the connection to the PROFIBUS subnetwork by selecting the module and then making your changes with the "Properties" button on the "General" tab under EDIT → OBJECT PROPERTIES.

CP 342-5DP as DP master

If a CP 342-5DP is the DP master, place it in the configuration table of the station, select it and then EDIT → OBJECT PROPERTIES. Set "DP Master" on the "Mode" tab.

The "Addresses" tab shows the user data address occupied by the CP in the address area of the CPU. From the viewpoint of the master CPU, the CP 342-5DP is an "analog module" with a module starting address and 16 bytes of user data.

Only DP standard slaves, or DP S7 slaves that behave like DP standard slaves, can be connected to a CP 342-5DP as DP master. You can find the suitable DP slaves in the hardware catalog under "PROFIBUS DP" and "CP 342-5DP as DP master". Select the desired slave type and drag it to the DP master system.

The transfer memory as DP master has a maximum length of 240 bytes. It is transferred as one with the loadable blocks FC 1 DP_SEND and FC 2 DP_RECV (included in the *Standard Library* under the *Communication Blocks* program).

The data consistency covers the entire transfer memory.

You read the diagnostics data of the connected DP slaves with FC 3 DP_DIAG (e.g. station list, diagnostics data of a specific station). FC 4 DP_CTRL transfers control jobs to the CP 342-5DP (e.g. SYNC/FREEZE command, CLEAR command, set operating state of the CP 342-5DP).

If you select CPU or CP 342-5DP, VIEW → ADDRESS OVERVIEW shows you a list of the assigned addresses, inputs and/or outputs. You can also screen the existing address gaps.

Configuring compact DP slaves

The compact DP slaves are to be found in the hardware catalog under "PROFIBUS DP" and the relevant sub-catalog, e.g. ET200B. Click on the DP slave selected and drag it to the icon for the DP master system.

You will see the properties sheet of the station; here, you set the node address and any diagnostics address. Then the DP slave appears as an icon in the upper section of the station window and the lower section contains a configuration table for this station.

A double-click on the icon in the upper section of the station window opens a dialog box with one or more tabs in which you set the desired station properties. In the lower sub-window, you then see the input/output addresses. Double-clicking on an address line shows you a window where you can change the suggested addresses.

The lower sub-window shows optionally the configuration table of the selected DP slave or of the master system (toggle with the "arrow" button).

Configuring modular DP slaves

The modular DP slaves can be found in the hardware catalog under "PROFIBUS DP" and the relevant sub-catalog, e.g. ET200M.

Click on the selected interface module (basic module) and drag it to the icon for the DP master system. This screens the properties sheet for the station; here, you set the node address and

any diagnostics address. Then the DP slave appears as an icon in the upper section of the station window and the lower section contains configuration table for this station.

Now place the modules that you can find in the hardware catalog *under the selected interface module (!)* in the configuration table. Double-clicking on the line opens the properties sheet of the module and allows you to parameterize the module.

The lower sub-window shows either the configuration table of the selected DP slave or of the master system (toggle with the "arrow" button).

Packing addresses

Digital electronics modules with two bit channels, e.g. ET200S or ET200pro, initially occupy one byte in the configuration table. After all modules have been configured, you can use the "Pack addresses" button to remove the gaps between the bit channels of the selected modules and thus occupy fewer addresses. The address areas for inputs, outputs and motor starters are packed separately.

Please note the following special features of a "packed" module:

- ▷ Slot assignment is no longer possible; the CPU cannot determine a geographical address for this module.
- ▷ No module status information for this module can be read.
- ▷ Interrupts cannot be assigned to a "packed" address. For this reason, a diagnostics address is assigned for this module (indicated by italics and a star in the configuration table). You can receive interrupt information under this address.
- ▷ "Pack addresses" and "Insert/remove module interrupt" are mutually exclusive.

Configuring a CPU with integral DP interface as an intelligent DP slave

With an appropriately equipped CPU, you can parameterize the station either as a DP master station or as a DP slave station. Before the station can be connected as a DP slave to a DP master system, it must be created. The procedure for doing this is exactly the same as that for a "normal" station; insert an S7 station into the project using the SIMATIC Manager and open the *Hardware* object. Drag a mounting rack to the window in the Hardware Configuration and place the desired modules. For configuring the DP slave, it is enough to place the CPU; you can add all other modules later.

When inserting the CPU, the properties sheet of the PROFIBUS interface is screened. Here, you must assign a subnetwork to the DP interface and you must assign an address. If the PROFIBUS subnetwork does not yet exist in the project, you can create a new one with the "New" button. This is the subnetwork to which the intelligent slave will later be connected.

You can open the properties sheet of the interface by selecting the DP interface and then EDIT → OBJECT PROPERTIES or by double-clicking on the interface. On the "Mode" tab, select the option "DP Slave". Now you can configure the user data interface on the "Configuration" tab from the viewpoint of the DP slave. Select MS (master/slave configuration) as the mode, and define the structure and addresses of the transfer interface from the viewpoint of the slave CPU (Figure 20.11). Chapter 20.4.1, "Addressing PROFIBUS DP" provides information on the user data interface under "Transfer memory on intelligent DP slaves".

The size and structure of the transfer memory is CPU-specific. On the CPU 315-2DP, for example, you can divide the entire transfer memory into up to 32 address areas that you can access separately. Such an address area can be up to 32 bytes in size. The entire transfer memory can have up to 244 input addresses and 244 output addresses.

The addresses defined here are located in the address volume of the slave CPU. These addresses must not overlap with addresses of centralized or distributed modules in the DP slave station. The lowest address of an address area is the "module starting address".

The user program in the slave CPU gets diagnostics information from the DP master via the diagnostics addresses specified on this tab.

You terminate configuration of the intelligent DP slave with STATION → SAVE AND COMPILE. Connecting the intelligent DP slave into the DP master system is described below.

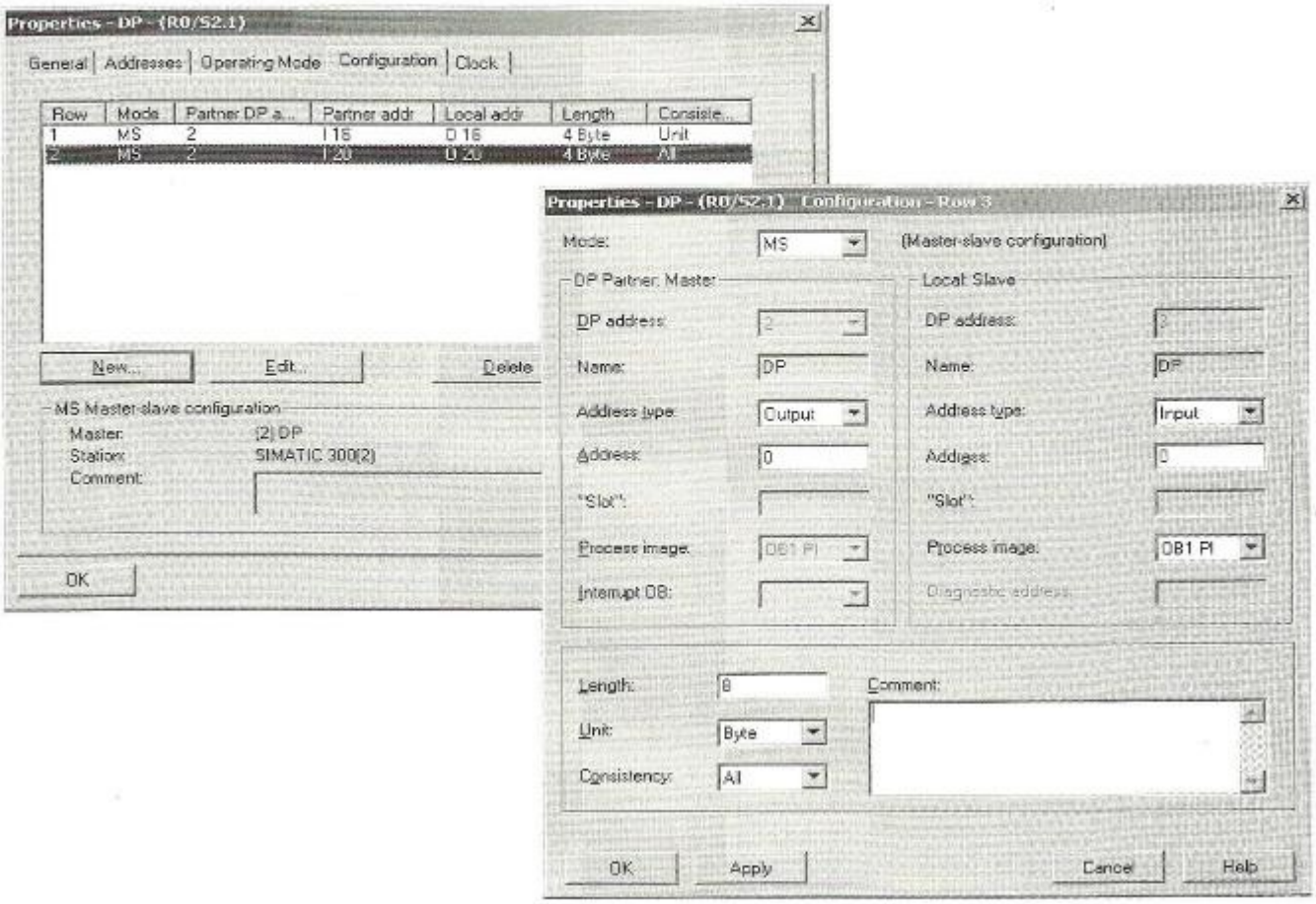


Figure 20.11 Configuring the Transfer Memory of an Intelligent Slave with Integral DP Interface

Configuring an ET200pro as an intelligent DP slave

Configuring an ET200pro station is very similar to configuring an S7-300 station. Insert a SIMATIC 300 station in the SIMATIC Manager under the project and open the *Hardware* object.

In the Hardware Configuration, drag the object *IM154-8 CPU* from the hardware catalog under "PROFIBUS DP" and "ET200pro" to the free window, or select it by double-clicking. On the displayed properties sheet of the Ethernet interface, set "not networked".

EDIT → OBJECT PROPERTIES in the case of a selected interface X1, or double-click the MPI/DP interface to open the Properties window. On the "General" tab, select PROFIBUS as the interface, and in the Properties window of the PROFIBUS interface select the station address and the PROFIBUS subnet. If the PROFIBUS subnet does not yet exist in the project, you can create one with the "New" button. This is the subnet to which the intelligent slave will later be connected.

On the "Operating Mode" tab, select the option "DP Slave". The meanings of the addresses on this tab and of the address on the "Addresses" tab are described in Chapter 20.4.1, "Addressing PROFIBUS DP" under , "Diagnostics address".

Now you can configure the user data interface on the "Configuration" tab from the viewpoint of the DP slave. Select MS (master-slave configuration) as the mode and define the structure of the addresses of the transfer interface from the viewpoint of the ET200pro CPU (Figure 20.11). You can find information on the user data interface in Chapter 20.4.1, "Addressing PROFIBUS DP" under, "Transfer memory on intelligent DP slaves".

In the case of the IM 154-8 CPU, you can divide the entire transfer memory into a total of 32 address areas that you can access separately. Such an address area can be up to 32 bytes in size. The entire transfer memory can encompass up to 244 input addresses and 244 output addresses.

The locally defined addresses are within the address volume of the ET200pro CPU. These addresses must not overlap addresses of the central or distributed modules in the ET200pro station. The lowest address of an address area is the "module starting address".

Further configuration of the ET200pro station is carried out in the same way as that for an S7-300 station with fixed slot addressing. You can only arrange the modules that can be found in the hardware catalog under "IM154-8 CPU".

You terminate configuration of the intelligent DP slave with STATION → SAVE AND COMPILE. Connecting the intelligent DP slave into the DP master system is described below.

Configuring an IM 151/CPU as an intelligent DP slave

If you want to generate an ET200S as an intelligent DP slave, first insert a SIMATIC 300 station under the project in the SIMATIC Manager and open the *Hardware* object.

Now drag the object *IM151/CPU* under "PROFIBUS DP" and "ET200S" in the Hardware Catalog to the free window or select it by double-clicking on it.

If the IM151/CPU does not have an MPI interface, the attributes window of the DP interface is displayed. Select the node address here and the PROFIBUS subnetwork (or generate one and assign it to the DP interface).

You will see a configuration table like the one for a SIMATIC 300 station. The intelligent IM 151 of the ET200S station is present here instead of the CPU.

Double-clicking on the IM 151 line opens the window for the IM properties; double-clicking on the DP interface opens the properties window of the interface. If you have not already done so, set the node addresses and the PROFIBUS subnetwork on the "General" tab (select "PROFIBUS" in the "Interface" box, and click the "Attributes" button).

Set the address ranges for the user data interface from the viewpoint of the DP slave at this point. The maximum size of the user data area is 32 bytes of inputs and 32 bytes of outputs for the IM 151/CPU. You can divide this area into eight sub-areas with different data consistency. The slave program receives diagnostics information from the DP master via the diagnostics address.

Further configuration of the ET200S station is carried out in the same way as for an S7-300 station with fixed slot addressing. You can only arrange the modules listed in the Hardware Catalog under "IM151/CPU".

Select STATION → SAVE AND COMPILE to conclude configuring the intelligent DP slave. Integration of the intelligent DP slave into the DP master system is described further below.

Configuring an S7-300 station with CP 342-5DP as an intelligent slave

If you insert an S7-300 station in the SIMATIC Manager, open the *Hardware* object and configure a "normal" S7-300 station. Among other things, you arrange a CP 342-5DP communications processor in the configuration table.

When inserting the station, the properties sheet of the DP interface appears; the subnetwork to which the intelligent DP slave is later to be connected is to be assigned to the DP interface here and you must also assign the node address.

To open the properties window, select the CP 342-5DP and then EDIT → OBJECT PROPERTIES, or double-click on the CP 342-5DP. Select the option "DP Slave" on the "Mode" tab.

The "Addresses" tab shows the user data interface from the viewpoint of the slave CPU (starting address and 16 bytes in length). The size of the transfer memory on the CP 342-5DP as DP slave is up to 86 bytes, and you can divide this into different address areas after connection to the master system.

STATION → SAVE AND COMPILE terminates configuration of the intelligent DP slave.

Connecting an intelligent DP slave to a DP master

You must have created a project and configured a DP master station and the intelligent DP slave (in each case at least with the DP interface). The DP master and the DP slave must be configured for the same PROFIBUS subnetwork.

Open the master station; a DP master system (broken black-and-white bar) must exist, if not, create it with INSERT → MASTER SYSTEM.

In the hardware catalog under "PROFIBUS DP" and "Configured Stations", you will find the objects that represent the intelligent slaves: "CPU 31x" or "CPU 41x" represents S7-300 or S7-400 stations with an integral DP slave, "ET200pro/CPU" and "ET200S/CPU" represent stations which were configured as DP slaves, and "S7-300 CP342-5 DP" represents S7-300 stations with CP 342-5 as the DP slave interface module. Select the desired slave type and drag it to the DP master system.

CPU, ET200pro or ET200S as DP slave

Dragging to a DP master system or double-clicking on the DP slave opens the properties sheet. The slaves already configured for this PROFIBUS subnetwork are listed on the "Connection" tab. Select the desired slave and click on the "Connect" button. This causes the active connection to be carried out at the bottom of the same dialog box.

On the "General" tab, you set the diagnostics address of the DP slave from the viewpoint of the master station.

On the "Configuration" tab, you now set the addresses of the user data interface from the viewpoint of the DP master. The output addresses on the master are the input addresses of the slave and vice versa. Chapter 20.4.1, "Addressing PROFIBUS DP" contains more information on the user data interface under "Transfer memory on intelligent DP slaves".

CP 342-5DP as DP slave

You open the properties sheet by dragging to the DP master system or by double-clicking on the DP slave. The slaves already configured for this PROFIBUS subnetwork are listed on the "Connection" tab. Select the desired slave and click on the "Connect" button. The active connection is then shown further down in the same dialog box.

When the DP slave is selected, its configuration table is shown in the lower section of the station window. Now you structure the transfer memory: select the "Universal submodule" from the hardware catalog (under the CP used), drag it to a line on the configuration table, or select a line and double-click on the "Universal submodule". You place one universal submodule for each individual (consistent) address area in the transfer memory; the maximum number is 32.

To open a window in which you define the properties of the address area, select the universal submodule and then EDIT → OBJECT PROPERTIES, or double-click on the line in the table: space, input or output area or both. Determine the starting address and the length of the area.

The addresses defined here are located in the address area of the master CPU. An area can be up to 64 bytes; the maximum overall size of the transfer area is 86 bytes.

If a CP 342-5DP is the DP master, structuring of the transfer memory can be omitted because the CP 342-5DP transfers the entire transfer area in one piece.

When dividing the transfer memory, you arrange the address areas together without gaps starting from byte 0. You access the entire assigned transfer memory in the slave CPU with the loadable blocks FC 1 DP_SEND and FC 2 DP_RECV (included in the *Standard Library* under the *Communication Blocks* program).

The data consistency covers the entire transfer memory.

On the "General" tab, you set the diagnostics address of the DP slave from the viewpoint of the master station. The diagnostics data are read with FC 3 DP_DIAG (in the master station).

Chapter 20.4.1, "Addressing PROFIBUS DP" contains more information on the user data interface under "Transfer memory on intelligent DP slaves".

Configuring the DP/DP coupler

The DP/DP coupler connects two PROFIBUS subnets. It is configured as a modular DP slave in each of the two subnets.

Prerequisite: Both subnets, each with a DP master system, must be configured. Open one of the stations with the DP master. In the hardware catalog under "PROFIBUS DP" and "Network components", you will find the *DP/DP coupler, Release 2* that you can drag to the DP master system with the mouse.

The properties sheet of the PROFIBUS interface appears on which you set the node address. You set the diagnostics address and further parameters on the "Parameters" tab in the object properties of the DP/DP coupler.

With a selected DP/DP coupler, the configuration table for the transfer memory appears. Now "Connect" the desired modules listed in the hardware catalog under DP/DP coupler into the configuration table, without gaps and starting at slot 1. You can set the universal module to the desired number of inputs and outputs. The user data addresses that you specify in the module properties are in the address area of the opened DP master CPU.

Configure the second part of the DP/DP coupler in the same way. The structure of the transfer memory must agree with that of the first part. Please note that inputs on one side are outputs on the other side, and vice versa. The addresses in both parts of the DP/DP coupler are oriented to the address assignments of the relevant master CPU and can differ from each other.

Configuring the DP/AS-i link

You configure the DP/AS-Interface link like a modular DP slave. You will find the modules that you can drag to the DP master system, such as the DP/AS-i link 20 below, in the hardware catalog under "PROFIBUS DP" and "DP/AS-i". Then you set the properties of the DP section and configure the AS-i slaves – depending on the link on the properties sheet or in the configuration table.

The AS-i master system with the AS-i slaves is not displayed as a subnet by the hardware configuration.

DP/AS-i Link 20

"Drag" the DP/AS-i Link 20 from the hardware catalog to the bar of the PROFIBUS DP master system. In the displayed windows, define first the preset configuration (16 or 20 bytes), and then the node address.

On the DP/AS-i link 20, you can define 16 bytes of inputs/outputs as the setpoint configuration, with an additional 4 bytes for control commands. In the latter case, the lower section of the window in the Hardware Configuration suggests 16 bytes of user data with addresses in the process image and 4 bytes of commands with addresses from, e.g. 512.

Select the DP slave and then EDIT → OBJECT PROPERTIES, or double-click on the DP slave, to open a window in which you can change the addresses suggested by the Hardware Configuration and also set the subprocess image, provided you have a suitable CPU.

Select the DP slave and then EDIT → OBJECT PROPERTIES, or double-click on the DP slave, to open the slave properties window. On the "Parameterize" tab, you set the parameters of the AS-i slaves, with 4 bits for each slave.

DP/AS-i Link Advanced

"Drag" the DP/AS-i Link Advanced – e.g. the double master 6GK1 415-2BA20 – from the hardware catalog to the bar of the PROFIBUS DP master system. In the displayed window, set the properties of the PROFIBUS interface. A configuration table with the AS-i interfaces and AS-i slaves for the two masters appears.

Now set the address area under which you want to access the AS-i slaves from the user program. EDIT→ OBJECT PROPERTIES with a selected AS-i master interface (in the configuration table), or double-click on the master line to open the properties dialog. On the "Digital Addresses" tab, set the start address and the area lengths reserved for inputs and outputs. This tab also contains the "Pack" and "Sort" buttons with which you can optimize the address assignments after you have configured the AS-i slaves.

Now "drag" the placeholder for an AS-i slave positioned under the link from the hardware

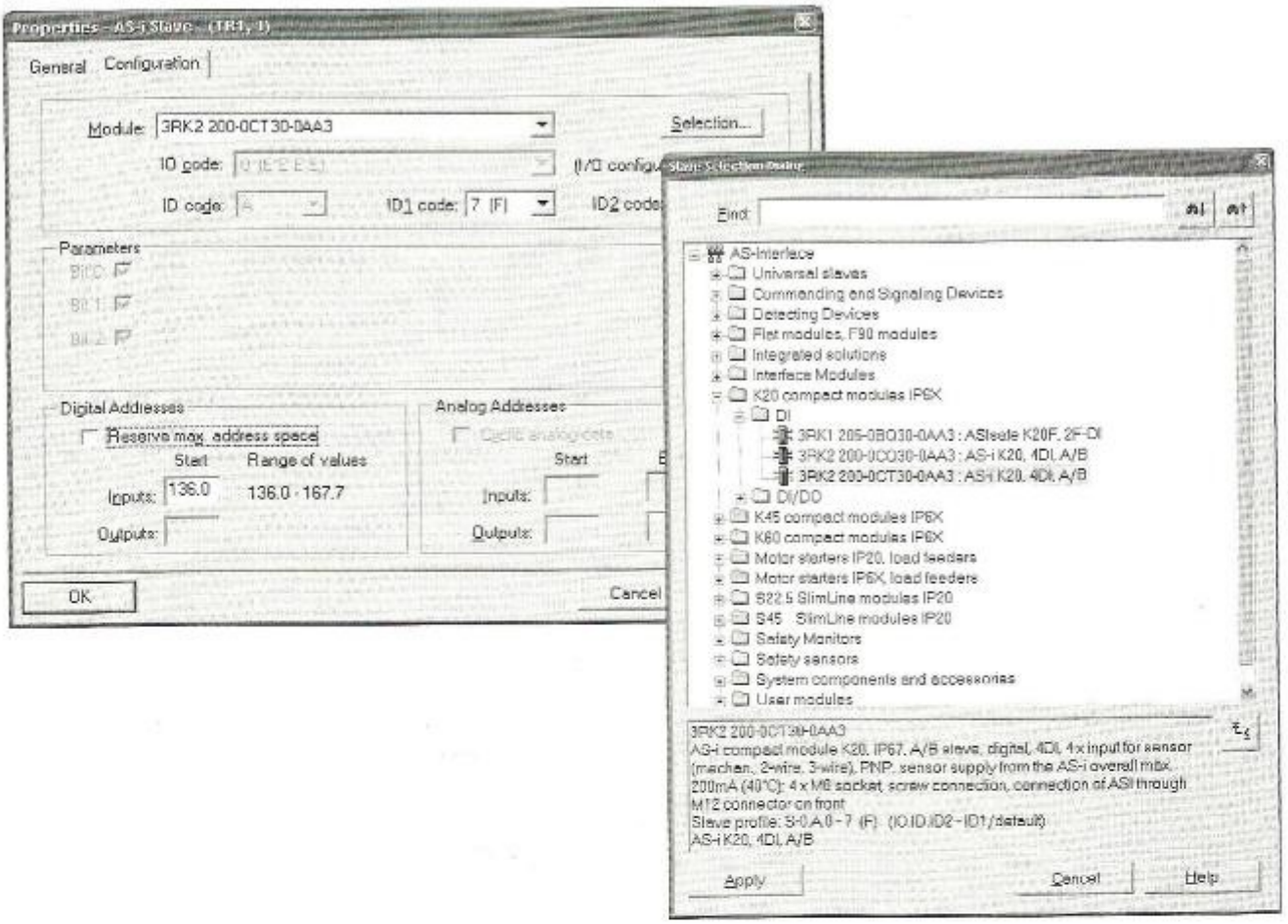


Figure 20.12 Properties and selection dialog for AS-i slaves in the case of DP/AS-i Link Advanced

catalog to the configuration table. Repeat this procedure for all scheduled AS-i slaves. EDIT→ OBJECT PROPERTIES with a selected AS-i slave (in the configuration table), or double-click on the slave line to open the properties dialog. Set the slave properties on the "Configuration" tab. Click "Selection" to view all the AS-i slaves known to the Hardware Configuration (Figure 20.12).

20.4.3 Special Functions for PROFIBUS DP

GSD files

You can "post install" DP slaves that are not included in the module catalog. For this purpose, you require the type file tailored to the slave (GSD file, General Station Description, device database file). From GSD version 3 onwards, DP slaves installed with a GSD file support the DPV1 functionality. Select OPTIONS → INSTALL GSD FILE in the Hardware Configuration and specify the directory of the GSD file or another STEP 7 project in the window that appears. STEP 7 accepts the GSD file and displays the slave in the hardware catalog under "PROFIBUS DP" and "Additional Field Devices".

STEP 7 saves the GSD files in the directory ...\Step7\S7DATA\GSD. The GSD files deleted when installing or importing at a later time are stored in the subdirectory ...\GSD\BKP*x*. From here, they can be restored with OPTIONS → INSTALL GSD FILE.

Configuring SYNC/FREEZE groups

The SYNC control command causes the DP slaves combined as a group to output their output states simultaneously (synchronously). The FREEZE control command causes the DP slaves combined as a group to "freeze" the cur-

rent input signal states simultaneously (synchronously), in order to allow them to be then fetched cyclically by the DP master. The UNSYNC and UNFREEZE control commands revoke the effect of SYNC and FREEZE respectively.

It is a requirement that the DP master and the DP slaves have the relevant functionality. From the object properties of a slave, you can see which command it supports (select DP slave, EDIT → OBJECT PROPERTIES, "General" tab under "SYNC/FREEZE Capabilities").

Per DP master system, you can form up to 8 SYNC/FREEZE groups that are to execute either the SYNC command or the FREEZE command or both. You can assign any DP slave to a group; on the CP 342-5DP of a specific version, one DP slave can be represented in up to 8 groups.

When you call SFC 11 DPSYC_FR, you cause the user program to output a command to a group (see Chapter 20.4.7, "System Blocks for Distributed I/O"). The DP master then sends the relevant command simultaneously to all DP slaves in the specified group.

You configure the SYNC/FREEZE group after configuring the DP master system (all DP slaves must exist in the DP master system). Select the DP master system (broken black-and-white bar) and select EDIT → OBJECT PROPERTIES. In the window that then appears, you first establish on the "Group Properties" tab the group commands to be executed (Figure 20.13), then you assign the DP slaves to the individual groups on the "Group Assignment" tab.

Here you select each of the DP slaves listed with its node number one after the other and select in each case the group to which it is to belong. If a DP slave cannot execute a specific command, such as FREEZE, the groups that

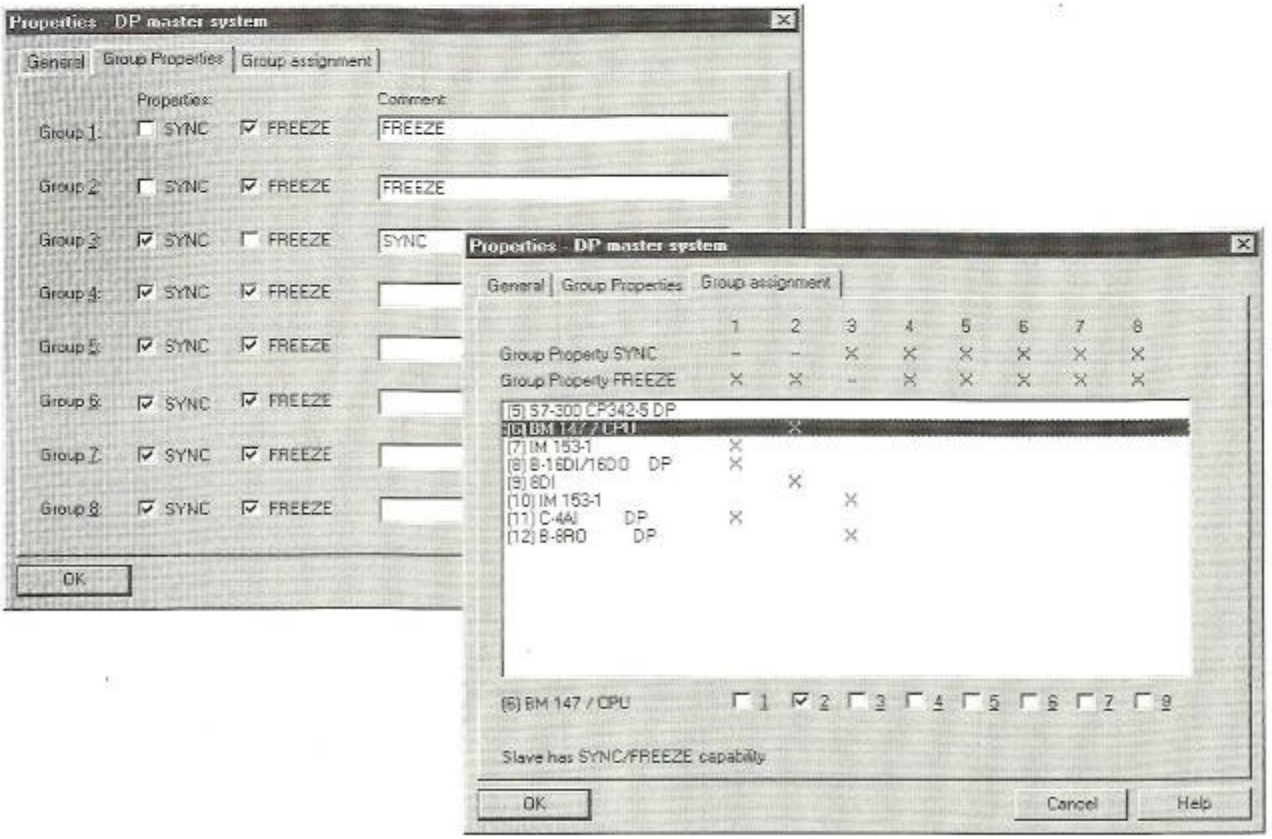


Figure 20.13 Configuring SYNC and FREEZE Groups

contain this command cannot be selected, e.g. all groups with the FREEZE command. Terminate configuring of the SYNC/FREEZE groups with OK.

Please note that when configuring bus cycles of the same length (equidistant), groups 7 and 8 acquire a special meaning.

Configuring PROFIBUS PA

For configuring a PROFIBUS PA master system and for parameterizing the PA field devices, use the Hardware Configuration tool (STEP 7 V5.1 SP3 and later) and the SIMATIC PDM optional software (with an earlier version of STEP 7). With the Hardware Configuration, you build up the connection to the DP master system with the DP/PA link: in the Hardware Catalog under "PROFIBUS DP" and "DP/PA Link", drag the IM 157 interface module to the DP master system. With the DP slave, a PA master system is simultaneously created in its own PROFIBUS subnetwork (45.45 kbits/s); it is indicated with a broken black-and-white bar.

The DP/PA coupler transfers the data unmodified and uninterpreted between the bus systems; for this reason, it is not parameterized. The PA field devices are addressed by the DP master. They can be incorporated as DP standard slaves into the Hardware Configuration of STEP 7 via a GSD file. Following this, you will find the PA field devices in the hardware catalog under "PROFIBUS DP" and "Other Field Devices".

Configuring direct data exchange (slave-to-slave communication)

In a DP master system, the DP master has exclusive control over the slaves assigned to it. With appropriately equipped stations, another node (master or intelligent slave, referred to as the "receiver" or "subscriber") can monitor the PROFIBUS subnetwork to learn which input data a DP slave ("sender" or "publisher") is sending to "its" master. This direct data exchange is also called "lateral communication". In principle all DP slaves from a specific revision level can function as senders in direct data exchange.

You configure direct data exchange with the Hardware Configuration in the Properties window of the DP slave (receiver) when all the stations on the PROFIBUS subnetwork are connected. Open the receiver station and select the DP interface and then EDIT → OBJECT PROPERTIES. The "Configuration" tab contains the transfer interface between the DP slave and the DP master. Click the "New" button, and set DX mode (direct data exchange) in the configuration window which is displayed. Also define the parameters for the DP partner (sender) in the same window.

You can also use direct data exchange between two DP master systems on the same PROFIBUS subnetwork. For example, the master in master system 1 can monitor the data of a slave in master system 2 in this way.

Configuring constant bus cycle times and isochrone mode

Equidistance

Normally, the DP master controls the DP slaves assigned to it cyclically without a pause. With S7 Communication, such as when the programming device executes modify functions via the PROFIBUS subnetwork, this can result in variations in the time intervals. If, for example, the outputs are to be modified via DP slaves at a regular interval, you can set constant bus cycles with the appropriately equipped DP master. For this purpose, the DP master must be the only Class 1 master on the PROFIBUS subnetwork. Constant bus cycle time behavior is possible with the bus profiles "DP" and "User-Defined". The PROFIBUS subnet must not be project-wide and no H system or CiR (Configuration in Run) object may be connected.

If you configure SYNC/FREEZE groups in addition to the constant bus cycle time, please note the following:

- ▷ For DP slaves in group 7, the DP master automatically initiates a SYNC/FREEZE command in every bus cycle. Initiation per user program is prevented.
- ▷ Group 8 is used for the constant bus cycle time signal and is disabled for DP slaves. You cannot configure constant bus cycle

time if you have already configured slaves for group 8.

Isochronous mode

Reference is made to isochrone mode if a program is executed in synchronism with the PROFIBUS DP cycle. In conjunction with constant bus cycle times, this results in reproducible response times of equal length to the I/O that include distributed signal acquisition, signal transfer via PROFIBUS and program execution including process image updating. The user program executed in isochrone mode is present in the organization blocks OB 61 to OB 64. The system functions SFC 126 SYNC_PI and SFC 127 SYNC_PO are available for updating of the process image in isochronous mode (see Chapter 21.8, "Synchronous Cycle Interrupts").

Figure 20.14 shows the timers involved in isochronous mode. *Ti* is the time required for reading in the process values. It contains the execution time in the input modules or electronics modules, and, in the case of modular DP slaves, the transfer time on the backplane bus. At the end of *Ti*, the input information for transfer using the global control command is available. Then the equidistance time begins. It is the time between two global control commands and encompasses the transfer to the subnet as well as the execution of the isochronous interrupt OB. Between completion of the execution of this OB to the next global control command there must be time for execution of the main program.

To is the time required for outputting the process values. It begins with the globalcontrol command and comprises the transfer time on the subnet as well as the processing time in the output modules or electronics modules. In the case of modular DP slaves, the transfer time on the backplane bus is also added. The response time in the case of isochronous mode is the total of the times *Ti*, equidistance time and *To*.

Correspondingly designed DP slaves allow a reduction in the response time thanks to *overlapping isochronous mode*. This involves overlapping updating of the input and output signals (overlapping of *Ti* and *To*). In this case, you must deactivate the checkbox "Times Ti and To identical for all slaves" on the "Equidistance" tab, and enter the individual times for the participating modules. If isochronous modules have both inputs and outputs, overlapping of *Ti* and *To* is not possible.

Configuring the isochronous mode

A prerequisite for configuration of the isochronous mode is the equidistant behavior and the corresponding functionality of the participating

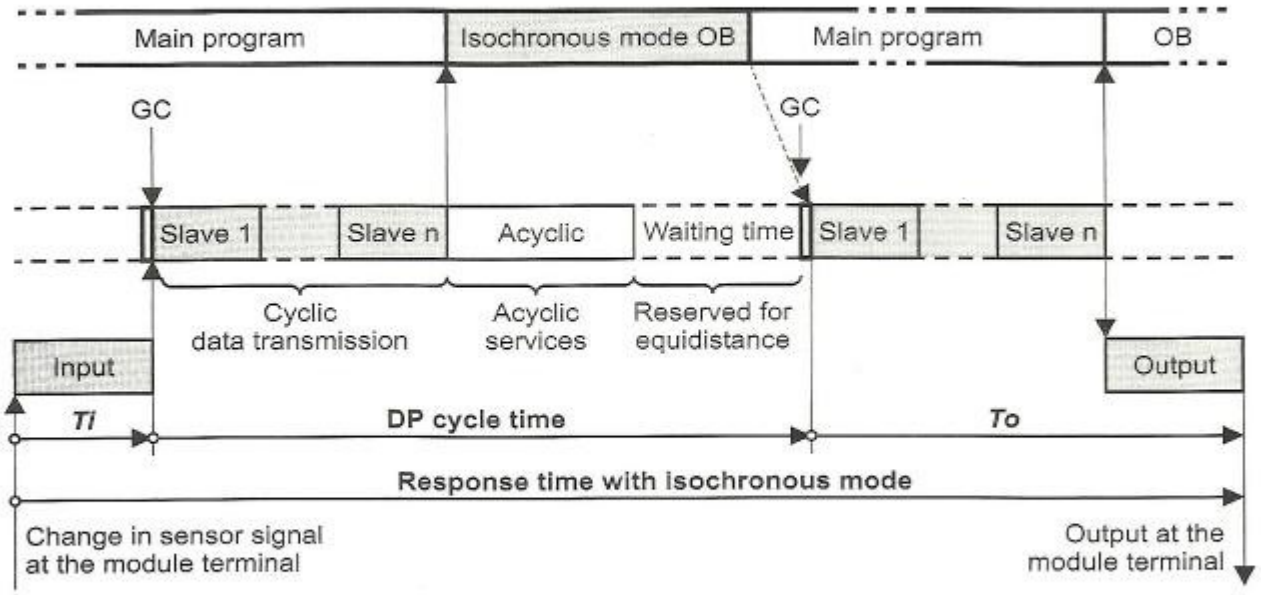


Figure 20.14 Response Time with Isochrone Mode and Constant Bus Cycle Times

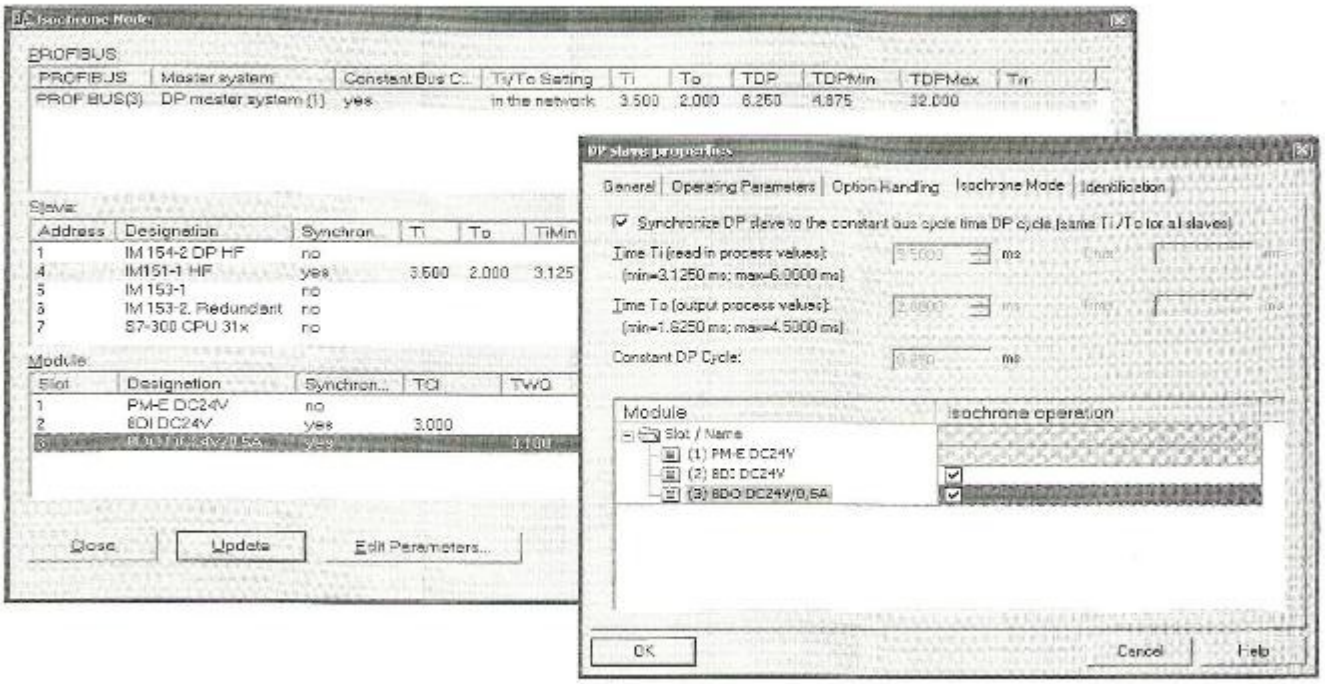


Figure 20.15 Isochronous mode: Overview of DP slave properties

DP components. Following configuration of the DP master system with appropriate modules (CPU with integral DP interface as well as ET200S and/or ET200M DP interfaces with the input/output modules), assign the priority, the DP master system and the partial process images to the organization block in the *CPU properties* on the "Synchronous cycle interrupts" tab.

To switch on the constant bus cycle times and isochrone mode, mark the *DP master interface* in the CPU and select EDIT → OBJECT PROPERTIES. In the displayed dialog box, click the "Properties" button in the "General" tab. In the properties window of the PROFIBUS interface, select the "Network settings" tab. Note that the constant bus cycle times properties can only be set with the bus profiles "DP" and "User-defined". Click the "Options" button and the selection box "Activate constant bus cycle times" in the options window which is then displayed. For isochronous mode, specify additionally the times *Ti* and *To*. Either select the "Times Ti and To same for all slaves" or set the times individually in the slave properties.

Each *module* participating in isochronous mode must have an address in the partial process image that has been assigned to the isochronous organization block in the CPU properties. Set the partial process image for the module in its properties dialog box on the "Addresses" tab.

The isochronous modules must be made known to the *DP interface module*. In the properties window of the DP slave, activate the option "Synchronize DP slave to constant bus cycle time DP cycle" on the "Isochrone mode" tab. Here you also select the modules or electronics modules involved in isochronous mode.

To *update Ti and To*, select the properties of the DP master system, then the (subnet) properties, and the "Options" button on the "Network settings" tab. When the "Recalculate" button is activated, STEP 7 updates all times involved in isochronous mode. You can modify the suggested equidistance time but not below the displayed minimum time. The "Details" button shows the individual proportions of the equidistance time. Please note that the equidistance time increases the more programming devices are connected directly to the PROFIBUS subnet and the more intelligent DP slaves are in the DP master system.

EDIT → ISOCHRONE MODE gives you an overview of all components involved in isochro-

nous mode and the relevant parameters (Figure 20.15). If the checkbox "Times Ti and To same for all slaves" under "Network settings" and "Options" has been deactivated, you can set the update times individually for each slave (prerequisite for overlapping isochronous mode). Select the DP slave in the "Isochrone mode" window. "Edit parameters" provides you with a dialog window for entering the individual update times and the modules involved in isochronous mode.

20.4.4 Addressing PROFINET IO

In a manner similar to how central modules are assigned to a CPU and controlled by it, the distributed modules with PROFINET IO (stations, IO devices) are assigned to an IO controller. The IO controller with all "its" IO devices is referred to as a PROFINET IO system.

Like central modules, the IO devices occupy addresses in the CPU's I/O area ("logical address area"). The IO controller is so to say "transparent" for the addresses of the IO devices; the CPU "sees" the addresses of the IO devices so that these addresses must not overlap with those of the central modules, not even with those of DP slaves in the DP master systems assigned to the CPU.

Each node operated on the Industrial Ethernet has an IP address. This is assigned during configuration of the IO controller. The IP addresses for the IO devices are derived from the IP address of the IO controller. In addition, an IO device is assigned a device name, a device number (node number), a geographical address (slot) and at least one diagnostics address (Figure 20.16).

MAC address

The MAC address is an address assigned to the device which is globally unambiguous. It con-

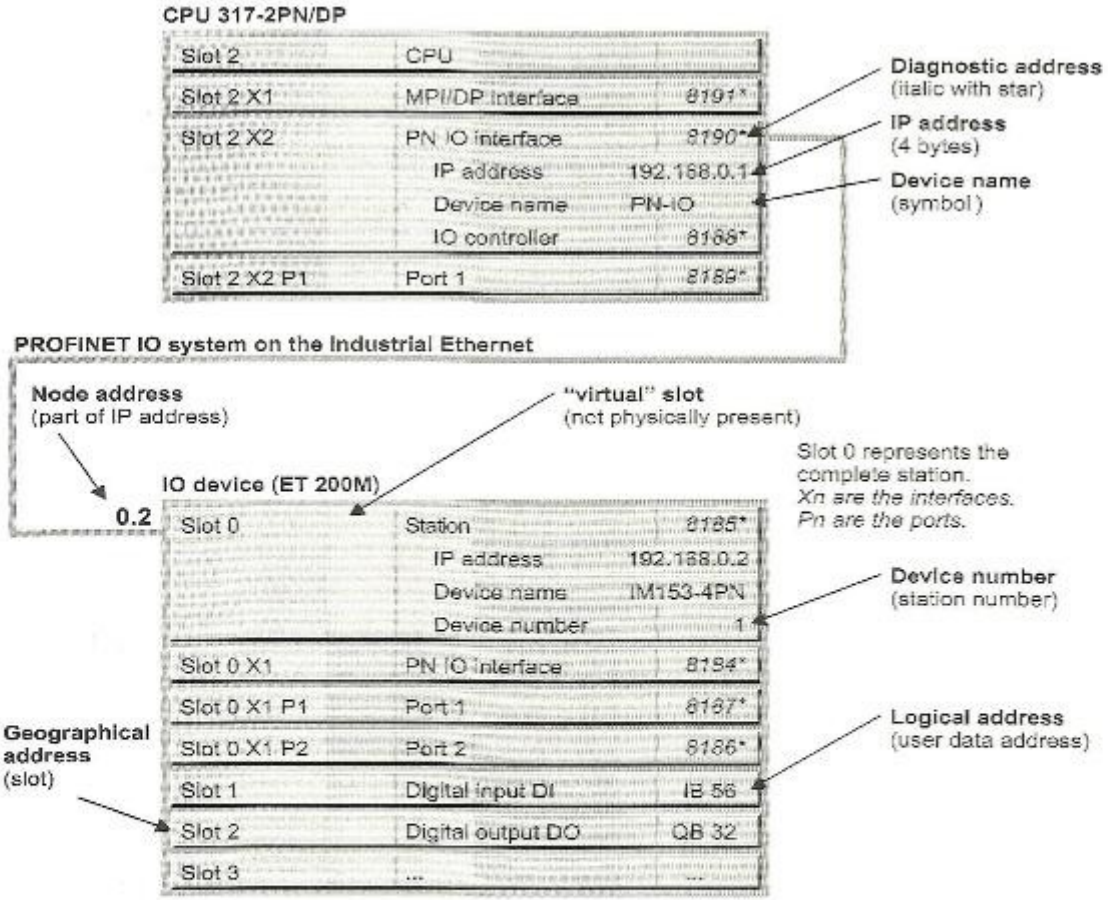
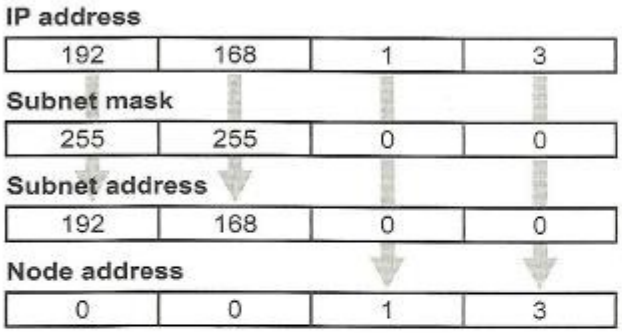


Figure 20.16 Addresses in a PROFINET IO system



The subnet address is present left-justified in the IP address and is generated by ANDing the IP address with the subnet mask.

Figure 20.17 Example of IP address structure

sists of three bytes with the vendor identification and three bytes with the device identification. The MAC address is usually printed on the device, and is assigned to it during the configuration procedure (if not already done in the factory).

IP address

Each node on the Industrial Ethernet subnetwork which uses the TCP/IP protocol requires an IP address. The IP address must be unambiguous on the subnetwork. It is assigned once to the IO controller for the nodes of a PROFINET IO system. Based on this, the Hardware Configuration assigns the IP addresses to the IO devices in ascending order.

The IP address is four bytes long, each being separated by a dot. Each byte is represented as a decimal number from 0 to 255.

The IP address comprises the address of the subnetwork and the address of the node. The subnetwork mask defines the share of the network address in the IP address. Like the IP address, it consists of four bytes which can assume a value of 255 or 0. Those bytes with a value of 255 in the subnetwork mask define the subnetwork address, the bytes with a value of 0 define the node address (Figure 20.17).

Device name, device number

During the configuration process, you assign the IO controller and each IO device a device name which must not be longer than 127 characters and consists of letters, digits, hyphens and dots.

The name of the IO system can be appended to the device name, separated by a dot. To do so, select "Use name in IO/Device/Controller" in the properties of the PROFINET IO system.

Supplementary to the device name, the Hardware Configuration also assigns a device number to each IO device which is independent of the IP address and which you can change. You can use this device number (station number) to address the IO device from the user program, e.g. as an actual parameter on a system block.

Geographical address

The geographical address identifies a module slot. With central modules, the geographical address contains the number of the rack and that of the slot. With PROFINET IO, the geographical address contains the number of the PROFINET IO system, the station number, the slot number and possibly a subslot number.

The "virtual" slot 0 (not physically present) represents the IO device. From slot 1 onwards are the user data and diagnostics data. The system functions SFC 70 GEO_LOG and SFC 71 LOG_GEO are available for conversion from the geographical address to the logical addresses and vice versa.

Logical address, module starting address

You use the logical address to access the user data of a station. Each byte of the user data is unambiguously identified by a logical address. The logical address corresponds to the absolute address; a symbol (name) can be assigned to it so that it is easier to read (symbolic addressing).

The smallest logical address of a module or station is the module starting address (see also Chapter 1.4, "Module Addresses").

Diagnostics address

Modules and stations which can deliver diagnostics data and do not have a user data address themselves are identified by the diagnostics address. The diagnostics address occupies one byte of peripheral input in the logical address volume. In the default setting, STEP 7 assigns the diagnostics address starting with the highest

address in the I/O area of the CPU. You can change the diagnostics address. The address overview in the Hardware Configuration tool identifies the diagnostics address by a star.

In the example shown (Figure 20.16), the MPI/DP interface of the CPU 317-2PN/DP is assigned diagnostics address 8191, the PN IO interface is assigned address 8190, and the IO controller is assigned address 8189. In the IO device, the interface (slot 0) is assigned diagnostics address 8188. The power module does not have its own user data, and is assigned the address 8187.

In the user program, the diagnostics data are scanned using system blocks. With a diagnostic interrupt, for example, the SFB 54 RALRM is used which reads the supplementary interrupt information. You can scan the diagnostics data record DS1 using the SFB 52 RDREC.

20.4.5 Configuring PROFINET IO

General procedure

You configure the distributed I/O an PROFINET IO essentially in the same way as the centralized modules. Instead of arranging modules in a mounting rack, you assign IO devices (nodes on the Industrial Ethernet subnetwork) to a PROFINET IO system. The following order is recommended for the necessary actions:

1) Create a new project or open an existing one with the SIMATIC Manager.
2) Create a Industrial Ethernet subnetwork in the project with the SIMATIC Manager.
3) Use the SIMATIC Manager to create the station in the project that is to accommodate the IO controller e.g. an S7-300 station.

   You start the Hardware Configuration by opening the station.
4) With the Hardware Configuration, you place an IO controller in the station. This can be, for example, a CPU with integral PN interface. You assign the previously created Ethernet subnetwork to the PN interface and you then have a PROFINET IO system. Import or modify the proposed device name and the IP address in the "General" tab. You can also configure the remaining modules later. Save and compile the station.
5) Using the mouse, drag the PROFINET nodes from the hardware catalog to the PROFINET IO system. Assign the device names, with the number (station number) if applicable.
6) If you use IRT communication, create a new sync domain, accept the participating PROFINET IO systems, and set the properties of the relevant devices.
7) If you use IRTtop, you must configure the network topology either in the Hardware Configuration directly at the port (interface connection) or centrally with the topology editor.
8) Save and compile all stations. The PROFINET IO system is now configured. You can now supplement the configuration with centralized modules, with DP master systems or further IO devices.

You can also represent the PROFINET IO system configured in this way graphically with the Network Configuration tool. Open Network Configuration by, for example, double-clicking on a subnetwork. Select VIEW → WITH DP SLAVES/IO DEVICES to display the IO devices. You can also create a PROFINET IO system (or more precisely, assign the nodes to an Ethernet subnetwork) with the Network Configuration tool. You parameterize the stations after opening them with the Hardware Configuration.

Before loading the configuration data, the device name must be assigned to each IO device ("naming"). In STOP mode, load the configuration data onto the CPU that accepts parameters such as the IP address. With the Hardware Configuration, load the data of the currently opened station (PLC→ DOWNLOAD); with the Network Configuration, you can send the data to several stations, for example, with PLC→ DOWNLOAD TO CURRENT PROJECT → STATIONS ON THE SUBNET.

At startup, the CPU transfers the configuring information to the IO devices and monitors the parameterization. The IO devices also receive their IP address with the parameters. Following successful parameterization, the user data are

then exchanged cyclically between the IO controller and IO devices in RUN.

Configuring the IO controller

You must have created a project and an S7 station with the SIMATIC Manager. You open the S7 station and create a mounting rack (see Chapter 2.3, "Configuring Stations"). When positioning a CPU with PN interface, enter the IP addresses and the subnetwork mask in the properties window which is displayed, and network the interface to an existing or newly created Ethernet subnetwork. After clicking the "OK" button you will be provided with a PROFINET IO system in the station window (black/white dashed rail).

If there is no PROFINET IO system available (it may be that it is obscured behind an object or is outside the visible area), create one by selecting the PN IO interface in the configuration window and then selecting INSERT → PROFINET IO SYSTEM.

Mark the black/white rail, and select EDIT → OBJECT PROPERTIES. Assign a name and an IO system number (from 100 to 115) in the "General" tab of the properties window. You can also select here whether the system name is to be part of the device name for the IO controller and IO devices. You can access the S7 subnetwork ID using the "Properties..." button.

The "Update time" tab shows the time span calculated by STEP 7 within which the IO controller has exchanged all data with its IO devices.

Change the device name of the IO controller with the PN IO interface selected and EDIT → OBJECT PROPERTIES; with the "Properties" button you will be provided with a window for changing the IP address, the subnetwork mask and the connection to an Ethernet subnetwork.

Configuring the IO device

The IO devices can be found in the hardware catalog under "PROFINET IO" and the corresponding subcatalog, e.g. I/O.

Click on the required interface (basic module) and drag it using the mouse to the symbol for the PROFINET IO system. Double click on the IO device to obtain the properties sheet of the station on which you can set the device name and number. You can change the proposed IP address by clicking the "Ethernet..." button.

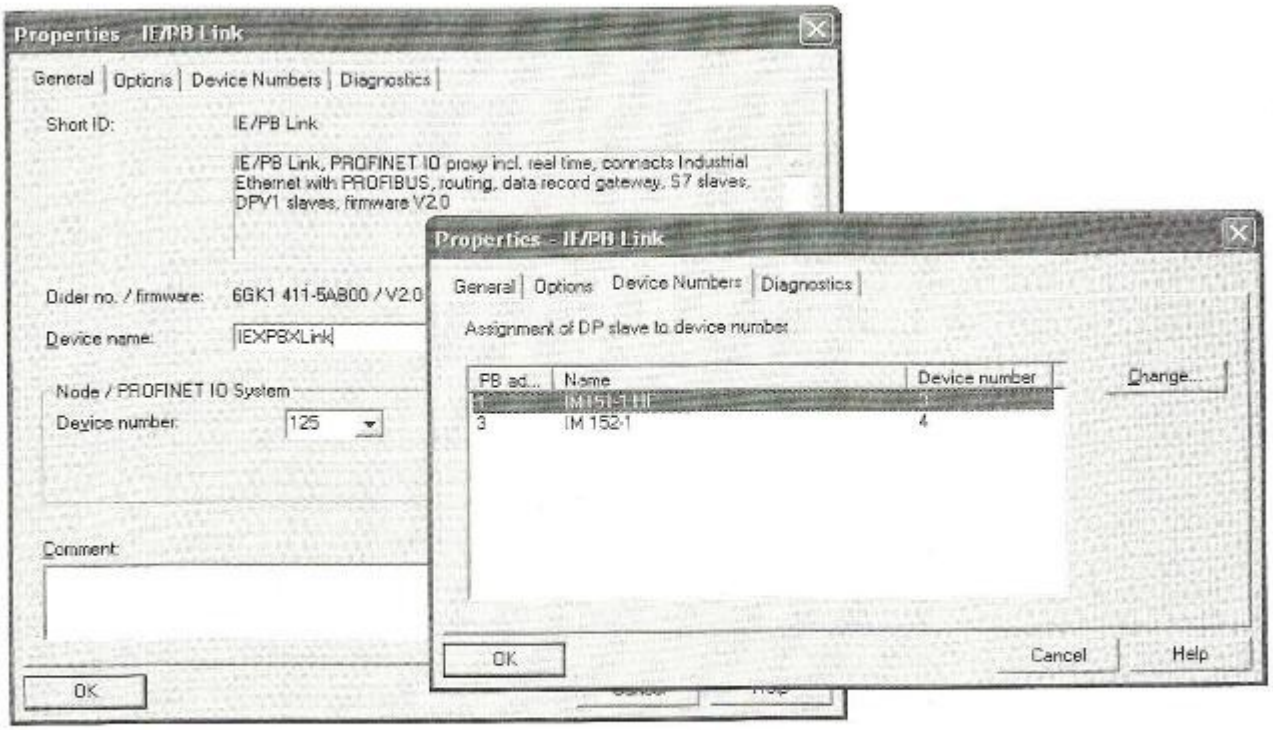


Figure 20.18 Configuring the PN/PN coupler: Connection and characteristics of the transfer interface

Now position the modules which you find in the hardware catalog *under the required interface (!)* in the configuration table. Double clicking on the line opens the properties sheet of the module and permits you to set the module parameters.

PN/PN coupler konfigurieren

The PN/PN coupler connects each of two Industrial Ethernet subnets to a PROFINET IO system. In each of the two PROFINET IO systems, one half of the PN/PN coupler appears as an IO device. With the configuration, you bring together the two IO devices and parameterize the PN/PN coupler.

Prerequisite: The two PROFINET IO systems are set up. The preferred method of configuring is to start with the "X1 side" of the PN/PN coupler. Open the relevant station (Figure 20.18).

You will find the symbol of the PN/PN coupler in the hardware catalog under "PROFINET IO" and "Gateway". Double-clicking on the symbol when the PROFINET IO system is already selected accepts the PN/PN coupler into the configuration. EDIT→ OBJECT PROPERTIES with the PN/PN coupler selected shows its properties window with the facility for setting the device name and the device number on the Ethernet subnet. You can change the IP address with the "Ethernet..." button.

In the configuration table that is displayed when the PN/PN coupler is selected, the station is located in the first line in slot 0. In the object properties of the station, select the diagnostics events to be reported on the "Parameters" tab. The diagnostics address of the station is on the "Addresses" tab.

On the "Connection" tab, set whether the "X1" or the "X2" side of the coupler is currently configured. If you have started with the "X1 side" of the coupler, and if both subnets to be connected are located in one project, you do not need to set anything else on the "Connection" tab.

In the object properties of the interface on slot X1, the "Addresses" tab contains the diagnostics address of the interface, and the "IO Cycle" tab contains information about the update time.

To configure the transfer memory, position the universal modules (found in the catalog under the PN/PN coupler) without gaps in the configuration table starting from Slot 1. In the properties of the universal module, set the I/O type (input area or output area or both), the relevant start address, and the area length, as well as the assignment to the process image. You can divide the transfer memory into a total of 16 areas whose data are consistently transferred. Save the previously completed settings with STATION → SAVE.

To configure the second side of the PN/PN coupler, open the station on the other Ethernet subnet and position the PN/PN coupler from the hardware catalog on the PROFINET IO system. Open Slot 0 of the PN/PN coupler, and set the connection partner (the already configured section of the coupler) on the "Connection" tab. If the subnet is in the same project, the structure of the transfer memory is transferred consistently. Now you can adapt the addresses according to your specifications and set the remaining parameters of the coupler.

If the two subnets are located in different projects, you must structure the transfer memory manually in both cases. Please note that an input on one side must correspond to an output on the other side, and vice versa. With different projects, you must assign the device names for both sides of the PN/PN coupler online.

On the "Connection" tab, you can also clear down the connection between the subnets again (enter "- - - - -" under "Subnet").

Configuring the IE/PB link

You can find the IE/PB link PNIO in the hardware catalog under "PROFINET IO" and "Gateway". Double clicking on the symbol with the PROFINET IO system previously selected outputs the properties window in which you can set the connection to a PROFIBUS subnetwork and the node number in this subnetwork in the "Parameters" tab.

EDIT → OBJECT PROPERTIES with the IE/PB link selected displays its properties window with the facility for setting the device name and number on the Ethernet subnetwork. You can change the IP address using the "Ethernet..." button.

The IE/PB link is quasi the DP master for the "subordinate" DP master system. You position the DP slaves from the hardware catalog on this master system, and assign them with the desired properties (see Chapter 20.4.2, "Configuring PROFIBUS DP").

The DP slaves require a device number in order to address them over PROFINET IO. You can make the assignments between node number on the PROFIBUS and device number on the PROFINET in the properties window of the IE/PB link in the "Device numbers" tab. As standard, STEP 7 uses the PROFIBUS addresses as the device number (indicated by a star on the device number). Select a DP slave in the list, and click the "Change" button.

The IE/PB link is able to pass on time-of-day telegrams and parameterization data records. You can make the settings for this in the properties window in the "Options" tab.

Configuring the IE/AS-i link

You will find the symbol of the IE/AS-i link PNIO in the hardware catalog under "PROFINET IO", "Gateway" and "Single master" or "Double master". Double-clicking on the link symbol when the PROFINET IO system is already selected positions the link at the IO system bar.

EDIT→ OBJECT PROPERTIES with the IE/AS-i link selected shows its properties window with the facility for setting the device name and the device number on the Ethernet subnet. You can change the IP address with the "Ethernet..." button.

The IE/AS-i link effectively functions as the AS-i-Master. The AS-i master system with the AS-i slaves is not displayed as a subnet by the Hardware Configuration. If the IE/AS-i link is selected, you receive the configuration table in which you "plug in" the AS-i slaves that appear in the catalog under the link symbol.

The configuration of the AS-i master system is described in Chapter 20.4.2, "Configuring PROFIBUS DP" under "Configuring the DP/AS-i link".

20.4.6 Special Functions for PROFINET IO

GSD files

You can "post install" IO devices that are not included in the module catalog. For this purpose, you require the type file tailored to the IO device (GSD file, Generic Station Description). Suitable for IO devices are GSD files of version 5 or later in XML format (GSDML, Generic Station Description Markup Language).

Select OPTIONS → INSTALL GSD FILE in the Hardware Configuration and specify the directory of the GSD file or another STEP 7 project in the window that appears. STEP 7 accepts the GSD file and displays the IO device in the hardware catalog under "PROFINET IO" and "Additional Field Devices".

STEP 7 saves the GSD files in the directory ...\Step7\S7DATA\GSD. The GSD files deleted when installing or importing at a later time are stored in the subdirectory ...\GSD\BKP*x*. From here, they can be restored with OPTIONS → INSTALL GSD FILE.

Real-time communication with PROFINET

PROFINET IO offers several types of data transfer:

- Non-time-critical data such as configuration and diagnostics information is transferred acyclically with the TCP/IP communication standard.
- User data (input/output information) is exchanged cyclically between the IO controller and the IO device (real-time RT) within a defined time period – the update time.
- Time-critical user data, for motion control applications, for example, is transferred with hardware support isochronously (isochronous real time IRT).

A permanent communication channel is reserved on the Ethernet subnet for IRT communication. RT communication – cyclic data exchange between the IO controller and IO devices – and non-real-time TCP/IP communication take place in parallel within the update time. In this way, all three communication types can exist in parallel on the same subnet.

Send cycle time/update time

Cyclic data exchange is handled within a specific time frame, the send cycle time. STEP 7 calculates the send cycle time from the configuration information on the PROFINET IO system. The send cycle time is the shortest possible update time. This is the time period within which each IO device in the IO system has exchanged its user data with the IO controller. The actual update time for an IO device can be a multiple of the send cycle time. You can increase the update time manually, in order to reduce the bus load, for example.

The update time is the same length for all IO devices in the IO system as standard. Under certain circumstances, you can reduce the update time for individual IO devices if you increase the time for other devices whose user data can be exchanged non-time-critically.

You can configure the send cycle time (without IRT communication) centrally in the Properties dialog box of the PN interface on the "PROFINET" tab, or in the Properties dialog box of the PROFINET IO system on the "Update time" tab.

This tab also lists the IO devices with the update times. You can increase the time for an IO device by selecting it and clicking on the "Edit" button, or you can set the time in the Properties dialog box of the IO device on the "IO cycle" tab.

In addition to the update time, you can set the watchdog timer in the device properties. This is the product of the update time and the "Number of accepted update cycles with missing IO data".

If there is at least one synchronized device in the IO system, the send cycle time is determined by the sync master of the sync domain and can only be modified in the properties of the sync domain. Select the IO system in the Hardware Configuration or the subnet in the Network Configuration. Select EDIT → PROFINET IO → DOMAIN MANAGEMENT. It can then no longer be modified at another location.

Real time

Real time (RT) means that a system processes external events within a defined time. If it responds predictably, it is called deterministic. In RT communication, transfer takes place at a specific time (send cycle time) within a defined interval (update time). PROFINET IO allows the use of standard network components for RT communication.

If not all data to be exchanged is transferred within the planned time frame, due, for example, to the addition of new network components, some data is distributed to other send frequencies. This can result in an increase in the update time for individual IO devices.

Isochronous real time

Isochronous real time (IRT) is hardware-supported real-time communication designed, for example, for motion control applications. IRT message frames are deterministically transmitted via planned communication paths in a specified order. IRT communication therefore requires network components that support this planned data transmission.

Isochronous real time is available in two versions that differ in their behavior with regard to network expansions, for example:

- ▷ In the case of IRTflex (RT Class 2), the send cycle time is automatically stepped down. This means that the data is distributed across several send cycle times if all the data to be exchanged is not transferred within one send cycle time. This increases the update time for individual IO devices.
- ▷ In the case of IRTtop (RT Class 3), transfer of all IO data within the update time calculated by the system is guaranteed. If the configuration is changed later, the update time is not automatically adjusted. In the consistency check, STEP 7 draws your attention to the fact that the update time has to be adjusted manually.

To be able to configure IRT communication, set up a new sync domain (see below) and determine a sync master to handle the synchronized distribution of the IRT message frames to the sync slaves. IRTtop requires a topology configuration (see Topology Editor below) and thus a

defined structure that takes account of the transmission properties of the cables and the switches used.

Sync domain

A sync domain is a group of PROFINET IO nodes that exchange synchronized data with each other. One node (this can be an IO controller or an IO device) assumes the role of the sync master, and the others are the sync slaves.

A sync domain can contain several IO systems, but an IO system is always assigned entirely to one single sync domain. Several sync domains can exist on one Ethernet subnet.

When an IO system is configured, a special sync domain is automatically created: the *sync-domain-default*. All configured IO systems, IO controllers and IO devices are first located in the sync domain *syncdomain-default*.

Create a new sync domain for IRT communication, and transfer the IO system (from the *sync-domain-default*) to the new sync domain. Not all devices of an IO system have to be synchronized, or in other words, exchange data with IRT communication. When configuring, the unsynchronized nodes are first also managed in the sync domain; at runtime only the synchronized nodes remain in the sync domain.

Configuring a new sync domain

Prerequisite: You have configured the Ethernet subnet with one or more PROFINET IO systems. The nodes involved in IRT communication must also support this function (Figure 20.19).

To create a new sync domain, select the PROFINET IO system in the Hardware Configuration or the subnet in the Network Configuration, and select EDIT → PROFINET IO → DOMAIN MANAGEMENT. The "Domain Management" window that appears shows the sync domain *syncdomain-default* and all IO systems on the same subnet that are located in the sync domain.

Create a new sync domain with the "New" button, assign a name, and select an IO system with the "Add" button. If you want to add further IO systems, repeat the procedure. IO systems added to the new sync domain are no longer a component part of the sync domain *sync-domain-default*.

The IO systems of the sync domain and the bus nodes of the selected IO system are displayed. Now select a device, then "Properties Device", and set the synchronization type (sync master or sync slave) and the RT class (RT, IRTflex or ITRtop) in the parameters in the Properties window. Proceed appropriately with the other bus nodes. Only one device can be the sync master, and all other devices are sync slaves. A sync domain may contain either devices of the classes IRTflex and RT, or devices of the classes IRTtop and RT.

Select the send cycle time in the "Send cycle time" field, and select the proportion of bus communication reserved for IRT with the "Details" button. Confirm with "OK" to save the settings.

Topology Editor

The Topology Editor allows cabling configuring of devices on the Industrial Ethernet subnet. The logical connections between the PROFINET devices are configured with the configuring tools Hardware Configuration and Network Configuration. The Topology Editor is used to configure the physical connections with the properties Length and Cable type for determining the signal runtimes. Use of the Topology Editor is a prerequisite for using IRT communication (isochronous real-time).

The physical connections between devices on the Ethernet subnet are point-to-point connections. The connections on a PN interface are called ports. The Ethernet cable connects a device port with a port on the partner device.

To enable several nodes to communicate with each other, they are connected to a switch that has several connections (ports) and that distributes signals. There are also S7 devices featuring a PN interface that has two or more ports separated by an integral switch. With this interface you can cable communication devices in a linear bus topology without external ports.

You can configure the connection of two ports with the Hardware Configuration. In the configuration table, select the port and then EDIT → OBJECT PROPERTIES. On the "Topology" tab in the Properties window, you can now deter-

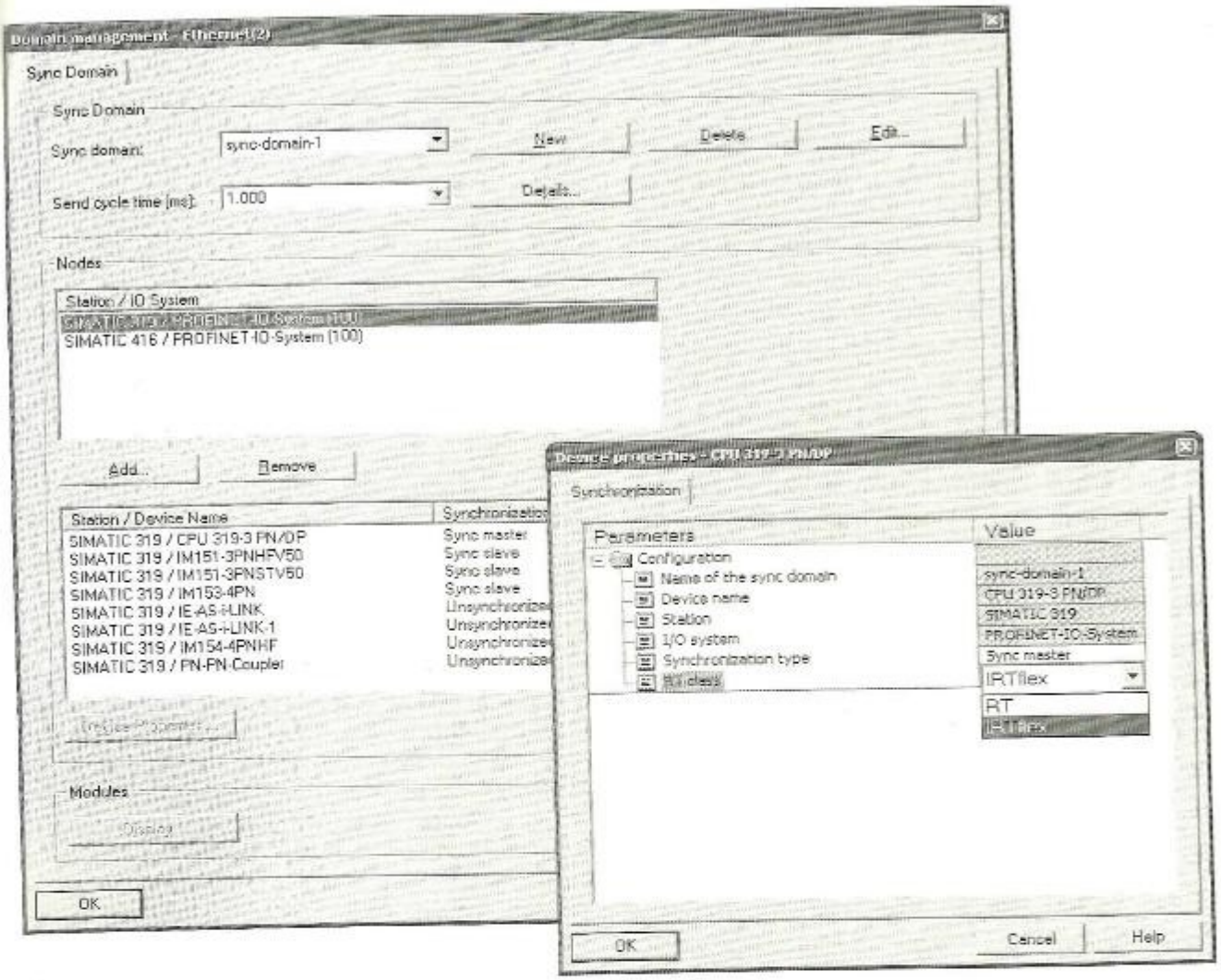


Figure 20.19 Configuring a new SYNC domain

mine the partner port and edit the properties of the cable.

A PN interface can have several ports. If an existing port is not configured as such (does not appear as a submodule in the STEP 7 environment) it is referred to as a *default port*. It is then not displayed in the configuration table, and any interconnections are not loaded onto the module. Default ports can only be handled with the Technology Editor.

Before you call the Topology Editor, use the Hardware Configuration or the Network Configuration to configure the communication partners on the Ethernet subnet including the necessary switches. In the Hardware Configuration, open the S7 station with the IO controller, select the PROFINET IO system and then select EDIT → PROFINET IO → TOPOLOGY. In the Network Configuration, select an Ethernet subnet and then EDIT → PROFINET IO → TOPOLOGY.

The *table view* shows the port pairs of all configured active and passive components in the interconnections table. With filter settings, you can display all ports, or only the interconnected ports, or only the non-interconnected ports (Figure 20.20).

You can set the connection to the partner port in the object properties of the port. You can clear down an interconnection by selecting the port and right-clicking on DISCONNECT PORT INTERCONNECTION .

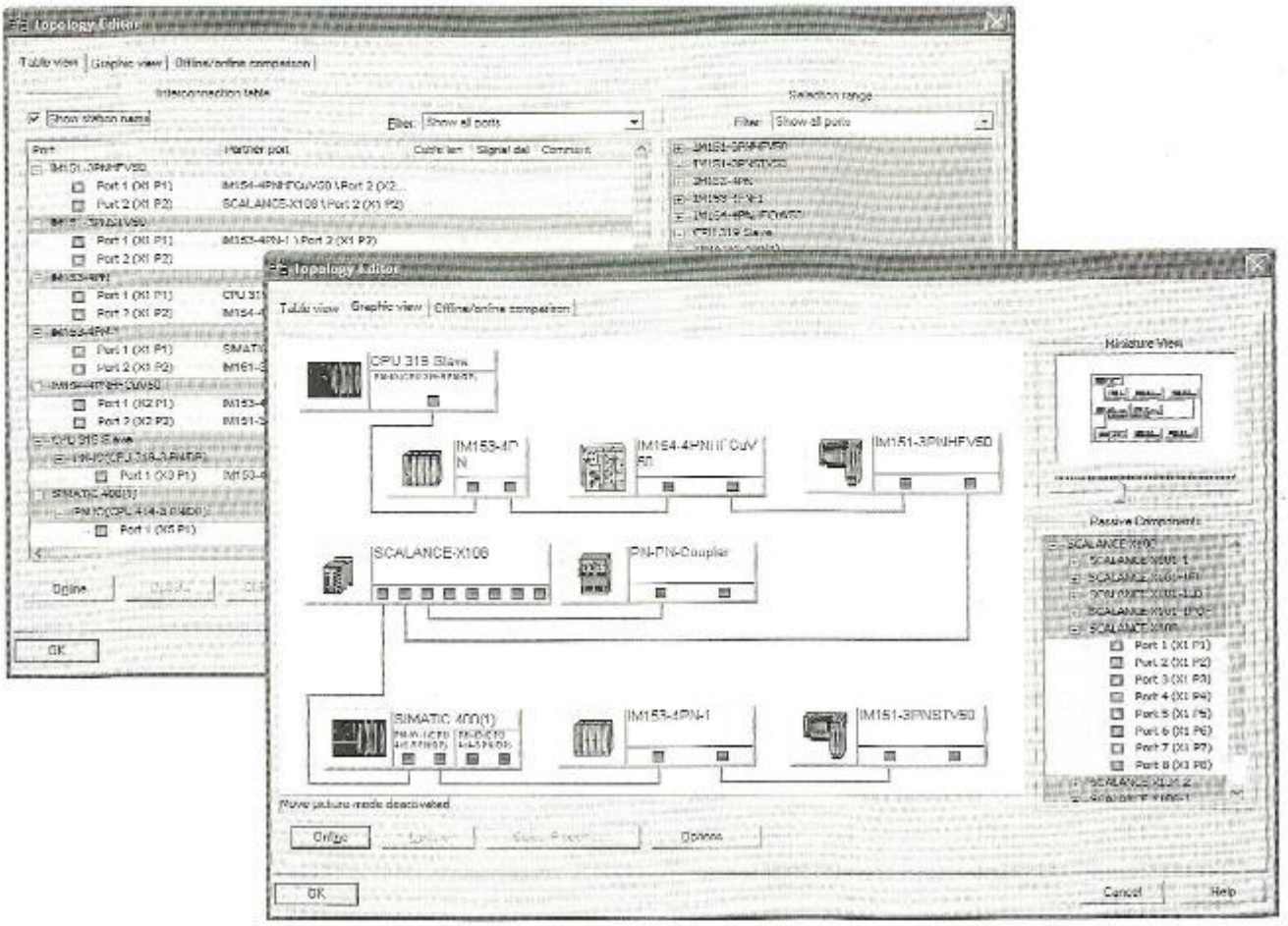


Figure 20.20 Tabular and graphical view of the Topology Editor

If there is a connection to the plant, you can use the "Online" button to check whether the devices configured offline are available and what their status is. The comparison is based on the device name, the IP address and the device ID. The data determined online is displayed in the columns "Status" and "Attenuation value".

In the *graphic view*, the topology window displays the devices, their ports and the interconnection. The configured devices are displayed in the offline view, and the actual devices available in the plant are displayed in the online view, provided there is an online connection to the plant.

To facilitate editing, you can "close" the representation of the stations, hide the thumbnail and the catalog of the passive components ("Options" button and "Options" tab), and enlarge or reduce the view by scrolling with the middle mouse key.

To interconnect two ports, select a port and drag a connection to the partner port while holding down the right mouse key. You can disconnect a port by selecting the connection with the right mouse key and then selecting DISCONNECT PORT INTERCONNECTION.

On the *Offline/Online comparison* tab, the topology configured offline is shown alongside the topology determined online for comparison purposes. All the stations and modules are displayed with their ports, relevant partner ports, and cable data. In this way, you can check the configuration with the connections and cables, and supplement missing system components if required.

A selection can be made with the relevant filter settings. In the overviews the determined differences, e.g. the modules that the Topology Editor could not assign, are highlighted in color. You can now undertake manual assignment.

20.4.7 System Blocks for Distributed I/O

You can use the following blocks in conjunction with the distributed I/O:

- FB 20 GETIO
  Read all inputs of a station
- FB 21 SETIO
  Write to all outputs of a station
- FB 22 GETIO_PA
  Read some inputs of a station
- FB 23 SETIO_PA
  Write to some outputs of a station
- SFB 75 SALRM
  Initiate interrupt
- SFC 7 DP_PRAL
  Initiate process interrupt
- SFC 11 DPSYN_FR
  Send SYNC/FREEZE commands
- SFC 12 D_ACT_DP
  Activate/deactivate distributed station
- SFC 13 DPNRM_DG
  Read diagnostics data from a DP standard slave
- SFC 14 DPRD_DAT
  Read user data
- SFC 15 DPWR_DAT
  Write user data
- SFC 103 DP_TOPOL
  Determine bus topology

You can find the parameters of the blocks in the Tables 20.9 (FB), 20.10 (SFB) and 20.11 (SFC).

Table 20.9 Function block parameters for accessing the distributed I/O

FB	Parameter	Declaration	Data Type	Contents, Description
20	ID	INPUT	DWORD	Logical user data start address
	STATUS	OUTPUT	DWORD	Error information of SFC 14 DPRD_DAT [1]
	LEN	OUTPUT	INT	Amount of bytes to be read
	INPUTS	IN_OUT	ANY	Destination area for the user data read (only data type BYTE permissible in the ANY pointer)
21	ID	INPUT	DWORD	Logical user data start address
	LEN	INPUT	INT	irrelevant
	STATUS	OUTPUT	DWORD	Error information of SFC 15 DPWR_DAT [1]
	OUTPUTS	IN_OUT	ANY	Source area for the user data to be written (only data type BYTE permissible in the ANY pointer)
22	ID	INPUT	DWORD	Logical user data start address
	OFFSET	INPUT	INT	Number of the first byte to be read (from number 0)
	LEN	INPUT	INT	Amount of bytes to be read
	STATUS	OUTPUT	DWORD	Error information of SFC 81 UBLKMOV [1]
	ERROR	OUTPUT	BOOL	Error occurred at signal state "1"
	INPUTS	IN_OUT	ANY	Destination area for the user data read (only data type BYTE permissible in the ANY pointer)
23	ID	INPUT	DWORD	Logical user data start address
	OFFSET	INPUT	INT	Number of the first byte to be written (from number 0)
	LEN	INPUT	INT	Amount of bytes to be written
	STATUS	OUTPUT	DWORD	Error information of SFC 81 UBLKMOV [1]
	ERROR	OUTPUT	BOOL	Error occurred at signal state "1"
	OUTPUTS	IN_OUT	ANY	Source area for the user data to be written (only data type BYTE permissible in the ANY pointer)

[1] Contains error information of the SFC used in the form DW#16#40xx xx00

The loadable function blocks FB 20 to FB 23 have interfaces compliant with PNO (PROFIBUS International) and can be used in conjunction with DP standard slaves and IO devices. You can find the blocks in the *Standard Library* supplied with STEP 7 in the program *Communication Blocks*.

You can use all listed system blocks with PROFIBUS DP, the system functions SFC 12, SFC 14 and SFC 15 are suitable for PROFINET IO. In order to read diagnostics data with PROFINET IO, use the SFB 54 RALRM.

With DPV1 mode set and DP slaves which support the DPV1 functionality, you can use other system blocks to parameterize and read the diagnostic data (see Chapters 21.9.3, "Reading additional Interrupt Information" and 22.5, "Parameterizing Modules").

FB 20 GETIO
Read all inputs of a station

FB 20 GETIO uses SFC 14 DPRD_DAT to consistently read all input data of a DP standard slave or an IO device or all data of an input area in the case of modular stations. The right-hand word of the ID parameter contains the start address of the input area to be read.

The destination area specified with the INPUTS parameter must be exactly the same length as the configured length of the input area read that is also output with the LEN parameter.

FB 21 SETIO
Write to all outputs of a station

FB 21 SETIO uses SFC 15 DPWR_DAT to consistently write all output data or all data of an output area of a DP standard slave or an IO device in the case of modular stations. The right-hand word of the ID parameter contains the start address of the output area to be written to.

The source area specified with the OUTPUTS parameter must be exactly the same length as the configured length of the output area to be written to. This is why information in the LEN parameter is irrelevant.

FB 22 GETIO_PA
Read some inputs of a station

FB 22 GETIO_PA uses SFC 81 UBLKMOV to consistently read some of the input data or some of the data of an input area of a DP standard slave or an IO device in the case of modular stations. The right-hand word of the ID pa-

Table 20.10 Parameters of the SFB 75 SALRM

Parameter	Declaration	Data Type	Contents, Description
REQ	INPUT	BOOL	Request to initiate with REQ = "1"
ID	INPUT	DWORD	Address of an address area in the transfer memory
ATYPE	INPUT	INT	Interrupt type: 1 = diagnostic interrupt 2 = process interrupt
ASPEC	INPUT	INT	Interrupt identification: 0 = no further information 1 = slot faulty (UP message) 2 = slot no longer faulty (DOWN message) 3 = slot still faulty (DOWN message)
LEN	INPUT	INT	Length in bytes (max. 16) of supplementary interrupt information to be sent
DONE	OUTPUT	BOOL	Interrupt has been transmitted when DONE = "1"
BUSY	OUTPUT	BOOL	Interrupt transmission still taking place when BUSY = "1"
ERROR	OUTPUT	BOOL	Default has occurred when ERROR = "1"
STATUS	OUTPUT	DWORD	Error information
AINFO	IN_OUT	ANY	Source area for supplementary interrupt information

Table 20.11 Parameters for SFCs Used to Reference the Distributed I/O

SFC	Parameter	Declaration	Data Type	Contents, Description
7	REQ	INPUT	BOOL	Request to initiate with REQ = "1"
	IOID	INPUT	BYTE	B#16#54 = input ID B#16#55 = output ID
	LADDR	INPUT	WORD	Starting address of an address area in the transfer memory
	AL_INFO	INPUT	DWORD	Interrupt ID (transfer of the start info of the interrupt OB)
	RET_VAL	RETURN	INT	Error information
	BUSY	OUTPUT	BOOL	No acknowledge yet from DP master when BUSY = "1"
11	REQ	INPUT	BOOL	Request to send with REQ = "1"
	LADDR	INPUT	WORD	Configured diagnostics address of the DP master
	GROUP	INPUT	BYTE	DP slave group (from the Hardware Configuration)
	MODE	INPUT	BYTE	Command (see text)
	RET_VAL	RETURN	INT	Error information
	BUSY	OUTPUT	BOOL	Job still running when BUSY = "1"
12	REQ	INPUT	BOOL	Request to activate/deactivate with REQ = "1"
	MODE	INPUT	BYTE	Function mode 0 Check whether the station is activated or deactivated 1 Activate station 2 Deactivate station
	LADDR	INPUT	WORD	Any logic address of the station
	RET_VAL	RETURN	INT	Result of check or error information
	BUSY	OUTPUT	BOOL	Job still running when BUSY = "1"
13	REQ	INPUT	BOOL	Read request with REQ = "1"
	LADDR	INPUT	WORD	Configured diagnostic address of the DP slave
	RET_VAL	RETURN	INT	Error information
	RECORD	OUTPUT	ANY	Destination area for the diagnostic data read
	BUSY	OUTPUT	BOOL	Read still in progress when BUSY = "1"
14	LADDR	INPUT	WORD	Configured start address (from the I area)
	RET_VAL	RETURN	INT	Error information
	RECORD	OUTPUT	ANY	Destination area for user data read
15	LADDR	INPUT	WORD	Configured start address (from the Q area)
	RECORD	INPUT	ANY	Source area for user data to be written
	RET_VAL	RETURN	INT	Error information
103	REQ	INPUT	BOOL	Triggering of topology determination with REQ = "1"
	R	INPUT	BOOL	Abort topology determination with R = "1"
	DP_ID	INPUT	INT	ID of DP master system whose topology is to be determined
	RET_VAL	RETURN	INT	Error information to the SFC
	BUSY	OUTPUT	BOOL	Determination still in progress when BUSY = "1"
	DPR	OUTPUT	BYTE	PROFIBUS address of diagnostics repeater signaling the fault
	DPRI	OUTPUT	BYTE	Test segment and error information of diagnostics repeater signaling the fault

rameter contains the start address of the input area, the OFFSET parameter contains the number of the first byte to be read, and the LEN parameter contains the number of bytes.

The input bytes to be read must be addressed in the process image of the input in order to use the FB 22 GETIO_PA. If possible, it is preferable to use a partial process image. Please ensure that the OFFSET and LEN parameters do not violate any boundaries with neighboring data of other stations.

If the destination area specified by the INPUTS parameter is smaller than the input area read, the function only transfers as many bytes as can be written to the destination area. If the destination area is larger, only the first LEN bytes of the area are written to. In both cases, no error is indicated on the ERROR parameter. ERROR only has signal state "1" if an error is reported when calling SFC 81 BLKMOV.

FB 23 SETIO_PA
Write some outputs to a station

FB 23 SETIO_PA uses SFC 81 UBLKMOV to consistently write some of the output data of an output area of a DP standard slave or an IO device or some of the data of an output area in the case of modular stations. The right-hand word of the ID parameter contains the start address of the output area, the OFFSET parameter contains the number of the first byte to be written, and the LEN parameter contains the number of bytes.

The output bytes to be written must be addressed in the process image of the output in order to use the FB 23 SETIO_PA. If possible, it is preferable to use a partial process image. Please ensure that the OFFSET and LEN parameters do not violate any boundaries with neighboring data of other stations.

If the source area specified by the OUTPUTS parameter is smaller than the output area to be written, the function only transfers as many bytes as the source area contains. If the source area is larger, only the first LEN bytes are transferred. In both cases, no error is indicated on the ERROR parameter. ERROR only has signal state "1" if an error is reported when calling SFC 81 BLKMOV.

SFB 75 SALRM
Initiate interrupt

With SFB 75 SALRM, you initiate a diagnostic interrupt or process interrupt in the DP master associated with an intelligent slave from the user program of that slave. You define the type of interrupt using the ATYPE parameter.

The interrupt request is initiated with REQ = "1"; the DONE, BUSY, ERROR and STATUS parameters indicate the job status. The job is complete (BUSY = "0") when the interrupt OB in the DP master has been executed.

The transfer memory between the DP master and the intelligent DP slave can be divided into individual address areas that represent individual modules from the viewpoint of the master CPU. You can initiate an interrupt in the master for each of these address areas ("virtual" modules). You specify the address area with the ID parameter which you occupy with a user data address from the viewpoint of the slave CPU. Bit 15 contains the I/O identification: "0" corresponds to an input address, "1" to an output address. The start information of the interrupt OB then contains the addresses of the "modules" initiating the interrupt from the viewpoint of the master CPU.

Use the AINFO parameter to transfer supplementary interrupt information which you have defined and which can be evaluated in the interrupt OB of the master CPU. The reference to AINFO is as an ANY pointer to a data area. The length of the sent information is defined by the LEN parameter and by the area length of the ANY pointer (the shorter length is decisive). The first 4 bytes are displayed in the start information of the interrupt OB in the master CPU in bytes 8 to 11 (the variable OBxx_POINT_ADDR for process interrupts, the data record DS 0 for the diagnostic interrupt). The complete supplementary interrupt information can be read in the master CPU using SFB 54 RALRM.

SFC 7 DP_PRAL
Initiate a process interrupt

With SFC 7 DP_PRAL, you initiate a process interrupt in the DP master associated with an

intelligent slave from the user program of that slave.

At the parameter AL_INFO you transfer an interrupt ID defined by you that is transferred to the start information of the interrupt OB called in the DP master (variable OBxx_POINT_ADDR). The interrupt request is initiated with REQ = "1"; the parameters RET_VAL and BUSY indicate the job status. The job is complete when the interrupt OB in the DP master has been executed.

The transfer memory between the DP master and the intelligent DP slave can be divided into individual address areas that represent individual modules from the viewpoint of the master CPU. The lowest address of an address area is taken as the module starting address. You can initiate a process interrupt in the master for each of these address areas ("virtual" modules).

You specify an address area at SFC 7 with the parameters IOID and LADDR from the viewpoint of the slave CPU (the I/O ID and the starting address of the slave side). The start information of the interrupt OB then contains the addresses of the "module" initiating the interrupt from the viewpoint of the master CPU.

SFC 11 DPSYN_FR
Send SYNC/FREEZE commands

With SFC 11 DPSYN_FR, you send the commands SYNC, UNSYNC, FREEZE and UNFREEZE to a SYNC/FREEZE group that you have configured with the Hardware Configuration. The SEND is initiated with REQ = "1" and is completed when BUSY = "0" is signaled.

In the parameter GROUP each group occupies one bit (from bit 0 = group 1 to bit 7 = group 8). The commands in the parameter MODE are also organized by bit:

- ▷ UNFREEZE, if bit 2 = "1"
- ▷ FREEZE, if bit 3 = "1"
- ▷ UNSYNC, if bit 4 = "1"
- ▷ SYNC, if bit 5 = "1"

In this manner you can use just one call to send several commands even to several groups.

In this way, SYNC mode and FREEZE mode on the DP slaves are first switched off. The inputs of the DP slaves are scanned in sequence by the DP master and the outputs of the DP slaves are modified; the DP slaves pass the received output signals immediately to the output terminals.

If you want to "freeze" the input signals of several DP slaves at a specific time, you output the command FREEZE to the relevant group. The input signals then read in sequence by the DP master have the signal states they had when "frozen". These input signals retain their value until you output another FREEZE command to cause the DP slaves to read in and hold the current input signals, or until you switch the DP slaves back to the "normal" mode with the UNFREEZE command.

If you want to output the output signals of several DP slaves synchronously at a specific time, first output the SYNC command to the relevant group. The addressed DP slaves then hold the current signals at the output terminals. Now you can transfer the desired signal states to the DP slaves. Following the transfer, you output the SYNC command again; this causes the DP slaves to switch the received output signals simultaneously through to the output terminals. The accessed DP slaves then hold the signals at the output terminals until you switch through the new output signals with a new SYNC command, or until you switch the DP slaves back to their "normal" mode with the UNSYNC command.

SFC 12 D_ACT_DP
Activate/deactivate distributed station

SFC 12 D_ACT_DP deactivates and activates stations of the distributed I/O and allows scanning of the deactivated or activated status. A distributed station can be a DP slave or an IO device.

SFC 12 D_ACT_DP is called in the cyclic program; calling in the restart routine is not supported. The SFC operates in asynchronous mode, i.e. processing of a job can extend over several program cycles. A "1" at the REQ parameter starts an activation or deactivation job. As long as the BUSY parameter is "1", the

REQ parameter must remain "1". The job has been finished if BUSY = "0".

After deactivation, a configured (and existing) station is no longer addressed by the DP master or the IO controller. The output terminals of deactivated output modules carry zero or a substitute value. The process image of the inputs of deactivated input modules is set to "0".

A deactivated slave can be taken off the bus without an error message; it will not be reported as faulty or missing. Calls of the asynchronous error organization blocks OB 85 (program execution errors, when the user data of the deactivated slave are located in an automatically updated process image) and OB 86 (station failure) cease. After deactivation, you must no longer access the DP slave from the program, since direct access will lead to an I/O access error with calling of the OB 122 or, when reading a data record with SFC 59 RD_REC or SFB 52 RDREC, the DP slave is signaled as being non-existent.

With SFC 12 D_ACT_DP, you reactivate a deactivated DP slave. The DP slave is configured and parameterized by the DP master or IO controller in the same way as during a station restore. On activation, the asynchronous error OBs 85 and 86 are not started. If the parameter BUSY has signal state "0" following activation, the DP slave can be accessed from the user program.

With a cold restart or warm restart, the CPU's operating system automatically activates the deactivated DP slave. An S7-300-CPU waits with start up until all stations are activated. An S7-400 CPU starts up and reports I/O access errors until the stations have been activated. With a hot restart, the DP slave status is retained.

SFC 13 DPMRM_DG
Read diagnostic data

SFC 13 DPNRM_DG reads diagnostic data from a DP slave. The read procedure is initiated with REQ = "1", and is terminated when BUSY = "0" is returned. Function value RET_VAL then contains the number of bytes read. Depending on the slave, diagnostic data may comprise from 6 to 240 bytes. If there are more than 240 bytes, the first 240 bytes are transferred and the relevant overflow bit is then set in the data.

The parameter RECORD writes to the area in which the read data are stored. Variables of data type ARRAY and STRUCT or an ANY pointer of data type BYTE (e.g. P#DBzDBXy.xBYTEnnn) are permissible as actual parameters.

Please note that the SFC 13 DPMRM_DG is an asynchronous system function. It must be processed until the BUSY parameter signals the status "0". The SFB 54 RALRM is available with newer CPUs which provides the data in synchronous mode, i.e. immediately following the call.

SFC 14 DPRD_DAT
Read user data

SFC 14 DPRD_DAT reads consistent user data with a length of 3 bytes or greater than 4 bytes from a DP slave or an IO device. You specify the length of the data consistency when you parameterize the station.

The parameter LADDR contains the module starting address of the user data (input area).

The parameter RECORD writes to the area in which the read data are stored. Variables of data type ARRAY and STRUCT or an ANY pointer of data type BYTE are permissible as actual parameters (e.g. P#DBzDBXy.xBYTEnnn).

SFC 15 DPWR_DAT
Write user data

SFC 15 DPWR_DAT writes consistent user data with a length of 3 bytes or greater than 4 bytes to a DP slave or an IO device. You specify the length of the data consistency when you parameterize the station.

The parameter LADDR contains the module starting address of the user data (output area).

The parameter RECORD writes to the area in which the read data are stored. Variables of data type ARRAY and STRUCT or an ANY pointer of data type BYTE are permissible as actual parameters (e.g. P#DBzDBXy.xBYTEnnn).

If peripheral outputs (PQ) are addressed whose addresses are in the process image of the outputs (PIQ), the process image is tracked as with

the transfer instruction (STL) or the MOVE box (LAD, FBD).

SFC 103 DP_TOPOL
Determine bus topology

SFC 103 DP_TOPOL uses diagnostics repeaters to determine the bus topology of the DP master system whose ID you specify in the DP_ID parameter. Determination is triggered by REQ = "1" and has been completed when BUSY = "0". You can abort determination of the topology using R = "1".

If an error which prevents determination of the bus topology is signaled by a diagnostics repeater, it is a shown in the DPR and DPRI parameters. If several diagnostics repeaters signal errors, the error message of the first one is displayed, and the complete diagnostics information can be read with SFC 13 DPNRM_DG.

The error information in the DPRI parameter is differentiated between temporary and permanent faults. Temporary faults, such as a loose contact, may be difficult to identify and may disappear on their own. You must eliminate permanent faults before you call the SFC 103 DP_TOPOL again to determine the topology.

After calling the SFC 103 DP_TOPOL, the determined data are available on the diagnostics repeater and can be read with the SFC 59 RD_REC or SFB 52 RDREC. The data comprise the topology of the bus segment (nodes and cable lengths), the contents of the segment diagnostics buffer (last ten events with error information, location and cause) and the statistics data (information on the quality of the bus system).

20.5 Global Data Communication

20.5.1 Fundamentals

Global data communication (GD communication) is a communications service integrated into the operating system of the CPU and used for exchanging small volumes of non-time-critical data via the MPI bus. The transferable global data include

- Inputs and outputs (process images)
- Memory bits
- Data in data blocks
- Timer and counter values as data to be sent.

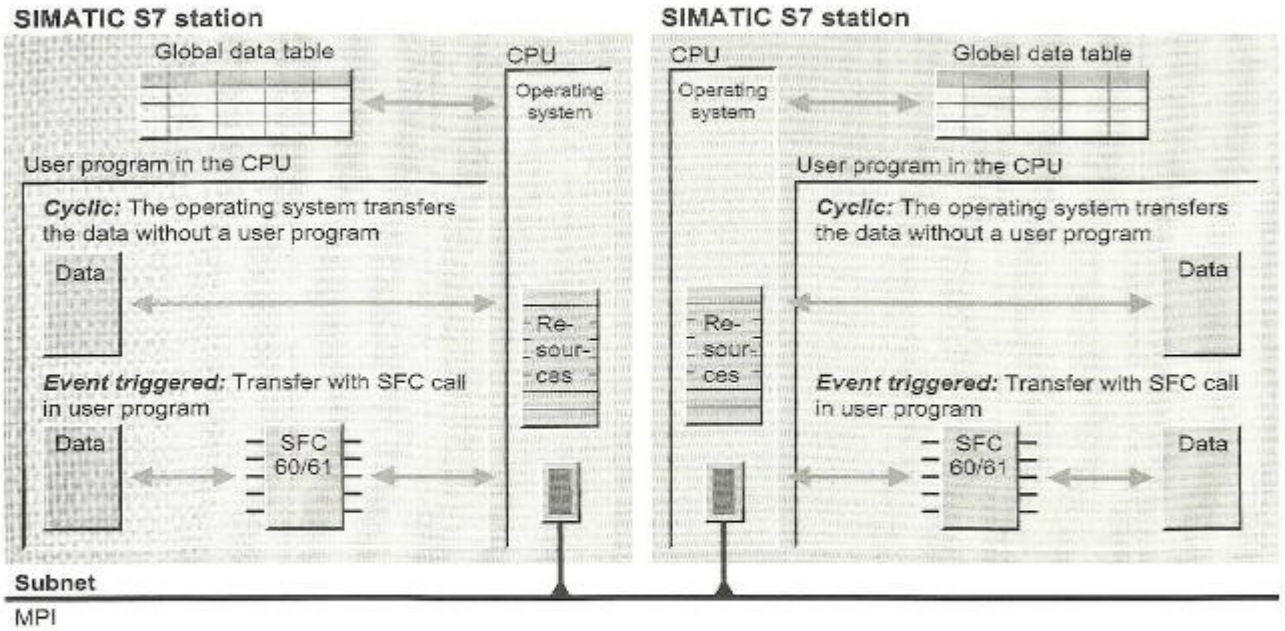


Figure 20.21 Global Data Communication

It is a requirement that the CPUs are networked together via the MPI interface or connected via the K bus as in the S7-400 mounting rack. All CPUs must exist in the same STEP 7 project in order to be able to configure GD communication.

The cyclic GD communication service does not require an operating system: there are system functions available for event-driven GD communication on the S7-400.

Please note that a receiver CPU does not acknowledge receipt of global data. The sender therefore does not receive any response to tell it if a receiver has received data and if so, which one. However, you can screen the communication status between two CPUs as well as the overall status of all GD circles of a CPU.

Sending and receiving global data is controlled with what are known as scan rates. These specify the number of (user program) cycles after which the CPU sends or receives the data. Sending and receiving takes place synchronously between the sender and the receiver at the cycle control point in each case, i.e. following cyclic program execution and before a new program cycle begins (like process image updating, for example).

Data is exchanged in the form of data packets (GD packets) between CPUs grouped into GD circles.

GD circle

The CPUs that exchange a shared GD packet form a GD circle. A GD circle can be any of the following

- The one-way connection of a CPU that sends a GD packet to several other CPUs that then receive that packet.
- The two-way connection between two CPUs where each of the two CPUs can send a GD packet to the other.
- The two-way connection between three CPUs where each of the three CPUs can send one GD packet to the other two CPUs (S7-400 CPUs only).

Up to 15 CPUs can exchange data with each other in one GD circle. One CPU can also belong to several GD circles. See Table 20.12 for the resources of each individual CPU here.

GD packet

A GD packet comprises the packet header and one or more global data elements (GD elements):

- Packet header (8 bytes)
- ID of 1st GD element (2 bytes)
- User data of 1st GD element (x bytes)
- ID of 2nd GD element (2 bytes)
- User data of 2nd GD element (x bytes)
- etc.

Each GD element consists of 2 bytes of description and the actual net data. 3 bytes are required

Table 20.12 CPU Resources for Global Data Communication

GD resources Max. number of:	CPU 312 CPU 313 CPU 314 CPU3xxC	CPU 315 CPU 317	CPU 412 CPU 414	CPU 416 CPU 417
GD circles per CPU	4	8	8	16
Receive GD packets per CPU	4	8	16	32
Receive GD packets per circle	1	1	2	2
Send GD packets per CPU	4	8	8	16
Send GD packets per circle	1	1	1	1
Max. size of a GD packet	22 bytes	22 bytes	64 bytes	64 bytes
Max. data consistency	22 bytes	22 bytes	1 variable	1 variable

in the GD packet to transfer a memory byte, 4 bytes are required for a memory word, and 6 bytes for a memory doubleword. A Boolean variable occupies 1 byte of net data; it therefore requires the same space as a byte-sized variable. Timer and counter values with 2 bytes each occupy 4 bytes each in the GD packet.

A GD element can also be an address area. MB 0:15, for example, represents the area from memory byte MB 0 to MB 15, and DB20.DBW14:8 represents the data area located in DB 20 that starts from data word DBW 14 and comprises 8 data words.

The maximum size of a GD packet is 32 bytes on the S7-300 and 64 bytes on the S7-400. The maximum number of net data bytes per packet is reached with the transfer of only one GD element that contains up to 22 bytes on the S7-300 and up to 54 bytes on the S7-400.

Data consistency

The data consistency covers one GD element. If a GD element overwrites a CPU-specific variable, the areas specified in Table 20.12 apply.

If a GD element is greater than the length of the data consistency, blocks with consistent data of the relevant length are formed, starting with the first byte.

20.5.2 Configuring GD communication

Requirements

You must have created a project, there must be an MPI subnetwork available and you must have configured the S7 stations. The CPU, at least, must be available in the stations. Under the "Properties" button of the MPI interface, on the "General" tab of the properties window of the CPU (double-click on the CPU line in the Hardware Configuration or on the line with the MPI interface submodule), you can set the MPI address and select the MPI subnetwork with which the CPU is connected.

Global data table

You configure GD communication by filling out a table. With the icon for the MPI subnetwork selected in the SIMATIC Manager or in the Network Configuration, you can call up an empty table with OPTIONS → DEFINE GLOBAL DATA. Select the station in the left half of the project selection window that then opens, and

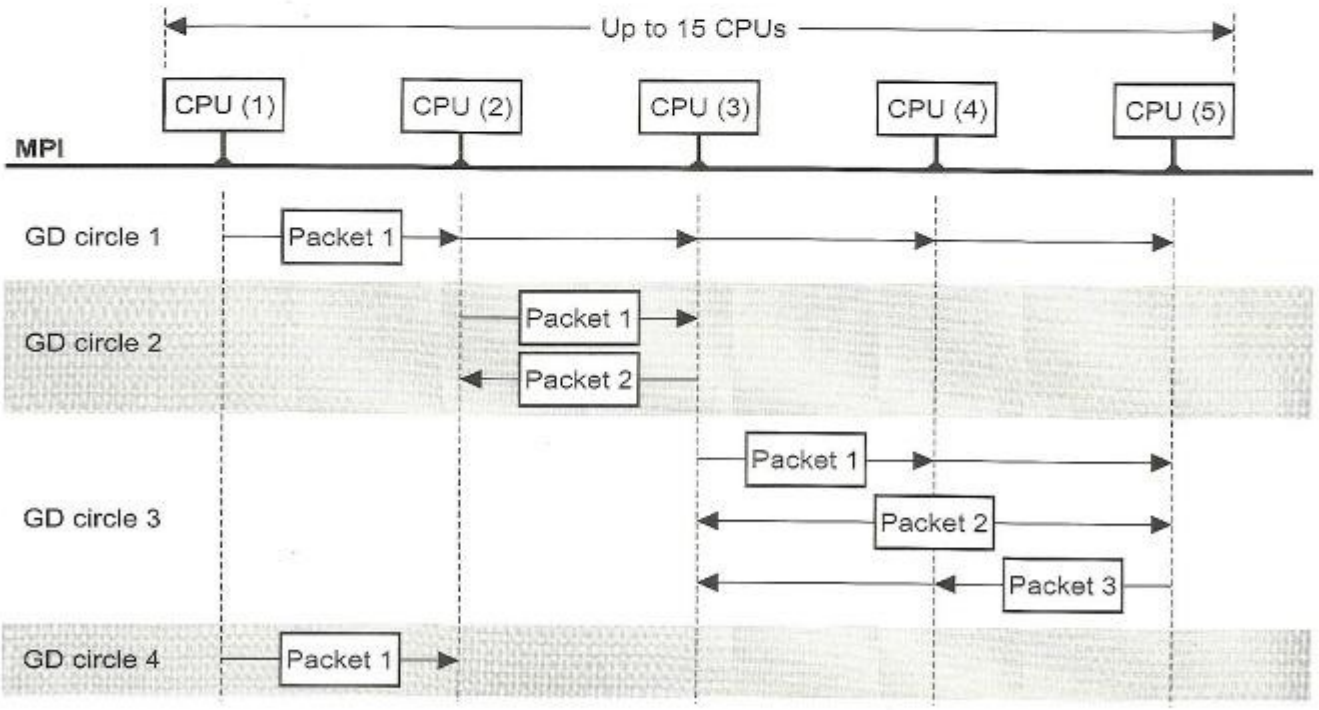


Figure 20.22 Example of GD Circles

select the CPU in the right half. This CPU is accepted into the global data table with "OK".

Proceed in exactly the same way with the other CPUs participating in GD communication. A global data table can contain up to 15 CPU columns.

To configure data transfer between CPUs, select the first line under the send CPU and specify the address whose value is to be transferred (terminate with RETURN).

With EDIT → SENDER, you define this value as the value to be sent, indicated by a prefixed character ">" and shading. In the same line under Receiver CPU, you enter the address that is to accept the value (the property "Receiver" is set as default). You may use timer and counter functions only as senders; the receiver must be an address of word width for each timer or counter function.

A line can contain several receivers but only one sender (Table 20.13). After filling this in, you select GD TABLE → COMPILE.

After compiling (phase 1), the system data created are sufficient for global data communication. If you also configure the GD status (the status of the GD connection) and the scan rates, you must then compile the GD table for a second time.

GD ID

Following error-free compiling, STEP 7 completes the "GD ID" column. The GD ID shows you how the transferred data are structured into GD circles, GD packets and GD elements. For example, the GD ID "GD 2.1.3" corresponds to GD circle 2, GD packet 1, GD element 3. You can then find the resource assignment (number of GD circles) per CPU in the CPU column of the global data table.

GD status

Following compiling, you can enter the addresses for the communication status into the global data table with VIEW → GD STATUS. The overall status (GST) shows the status of all communications connections in the table. The status (GDS) shows the status of a communications connection (a transmitted GD packet). The status is shown in a doubleword in each case.

Scan rates

The GD communication service requires a significant portion of execution time in the CPU operating system and demands transmission time on the MPI bus. To keep this "communications load" to a minimum, it is possible to specify a "scan rate". A scan rate specifies the number of program cycles after which the data (or more precisely, a GD packet) are to be sent or received.

Table 20.13 Example of a GD Table with Status and Scan Rates

GD Identifier	Station 417 \ CPU417 (3)	Station 417 \ CPU414 (4)	Station 416\ CPU 416 (5)	Station 315 Slave\ CPU315 (7)	Station 314CP\ CPU314 (10)
GST	MD100	MD100	MD100	DB10.DBD200	DB10.DBD200
GDS 1.1	DB9.DBD0		MD92	DB10.DBD204	DB10.DBD204
SR 1.1	44	0	44	8	8
GD 1.1.1	>DB9.DBW10		MW90	DB10.DBW208	DB10.DBW208
GDS 2.1	MD96	MD96			
SR 2.1	44	23	0	0	0
GD 2.1.1	>Z10:10	DB19.DBW20:10			
GDS 3.1			MD96		
SR 3.1	0	0	44	8	8
GD 3.1.1			>MW98	DB10.DBW220	DB10.DBW210

Since the data are not updated in every program cycle with a scan rate, you should avoid sending time-critical data via this form of communication.

After the first (error-free) compilation, you can use VIEW → SCAN RATES to define the scan rates (SRs) yourself for each GD packet and each CPU. The scan rate is set as standard in such a way that with an "empty" CPU (no user program) the GD packets are sent and received approximately every 10 ms. If a user program is then loaded, the time interval increases.

You can enter the scan rates in the area between 1 and 255. Please note, that as the scan rates decrease, the communications load on the CPU increases. To keep the communications load within tolerable limits, set the scan rate in the send CPU in such a way that the product of scan rate and cycle time on the S7-300 is greater than 60 ms and on the S7-400 greater than 10 ms. In the receive CPU, this product must be less than that in the send CPU to avoid the loss of any GD packets.

With scan rate 0, you switch off data exchange of the relevant GD packet if you only want to send or receive it event-driven with SFCs.

After configuring the GD status and the scan rates, you must compile the GD table for a second time. Then STEP 7 enters the compiled data in the *System data* object. GD communication becomes effective when you transfer the GD table to the connected CPUs with PLC → DOWNLOAD TO MODULE.

GD communication also becomes effective when the *System data* object, that contains all hardware settings and parameter settings, is transferred.

20.5.3 System Functions for GD Communication

In S7-400 systems, you can also control GD communication in your program. Additionally or alternatively to the cyclic transfer of global data, you can send or receive a GD packet with the following SFCs:

- SFC 60 GD_SND
  Send GD packet
- SFC 61 GD_RCV
  Receive GD packet

The parameters for these SFCs are listed in Table 20.14. The prerequisite for the use of these SFCs is a configured global data table. After compiling this table, STEP 7 shows you, in the "GD Identifier" column, the numbers of the GD circles and GD packets which you need for parameter assignments.

SFC 60 GD_SND enters the GD packet in the system memory of the CPU and initiates transfer; SFC 61 GD_RCV fetches the GD packet from the system memory. If a scan rate greater than 0 has been specified for the GD packet in the GD table, cyclic transfer also takes place.

If you want to ensure data consistency for the entire GD packet when transferring with SFCs 60 and 61, you must disable or delay higher-priority interrupts and asynchronous errors on both the Send and Receive side during processing of SFC 60 or SFC 61.

The SFCs need not be called in pairs; "mixed" operation is also possible. For example, you can use SFC 60 GD_SND to have event-driven transmission of GD packets but then receive cyclically.

Table 20.14 SFC Parameters for GD Communication

Parameter	Present in SFC		Declaration	Data Type	Contents, Description
CIRCLE_ID	60	61	INPUT	BYTE	Number of the GD circle
BLOCK_ID	60	61	INPUT	BYTE	Number of the GD packet to be sent or to be received
RET_VAL	60	61	RETURN	INT	Error information

20.6 S7 Basic Communication

20.6.1 Station-Internal S7 Basic Communication

Fundamentals

With station-internal S7 basic communication, you can exchange data between programmable modules within a SIMATIC station. The communication functions required here are SFCs in the operating system of the CPU. These SFCs establish the communication connections themselves, if necessary. For this reason, these station-internal connections are not configured via the connection table ("Communication via non-configured connections", SFC communication).

Station-internal S7 basic communication can take place, for example, parallel to cyclic data exchange via PROFIBUS DP between the master CPU and the slave CPU, with data transfer being event-driven (Figure 20.23).

Addressing the nodes, connections

Node identification is derived from the I/O address: at the LADDR parameter, you specify the module starting address and at the IOID parameter, you specify whether this address is in the input area or the output area.

These system functions establish the necessary communication connections dynamically and they clear the connections down again following completion of the job (programmable). If a connection buildup cannot be executed due to lack of resources either in the sending device or in the receiving device, "Temporary lack of resources" is signaled. Transfer must then be reinitiated. There can only be one connection between two communication partners in each direction.

You can use one system function for different communication connections by modifying the block parameters at runtime. An SFC cannot interrupt itself. A program section in which one of these SFCs is used can only be modified in the STOP mode; following this, a cold or warm restart must then be executed.

User data, data consistency

These SFCs transfer up to 76 bytes as user data. Regardless of the direction of transfer, the operating system of a CPU arranges the user data in blocks that are consistent within themselves. The length of consistent transferred data is a CPU-specific variable. If two CPUs exchange data, the block size of the "passive" CPU is decisive for data consistency.

Configuring station-internal S7 basic communication

Station-internal S7 basic communication is a special case in that it requires no configuring since data transfer is handled via dynamic connections. You simply use an existing PROFI-

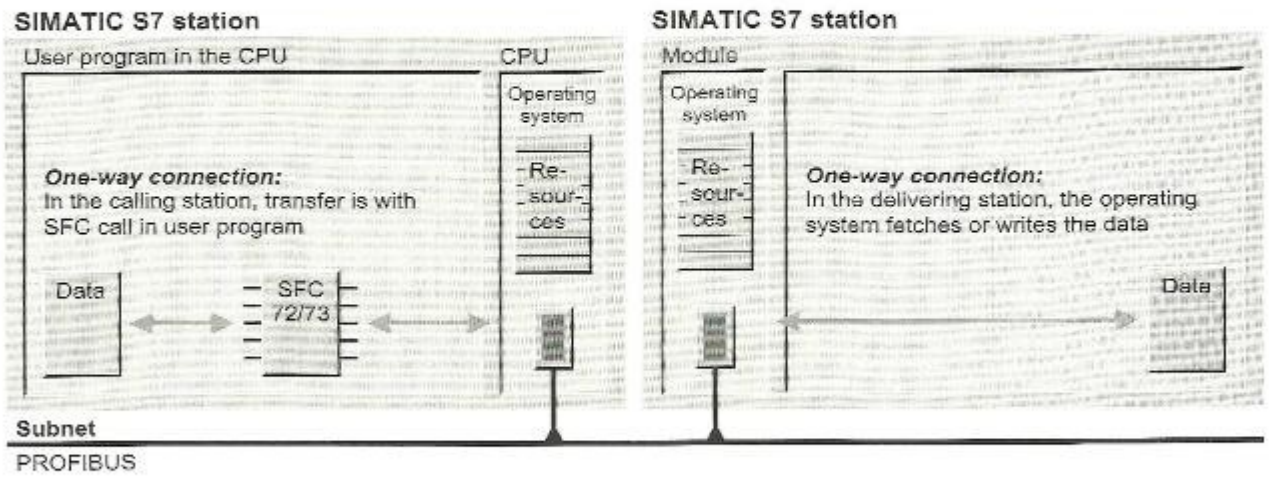


Figure 20.23 Station-Internal S7 Basic Communication

Table 20.15 SFC Parameters for Station-Internal S7 Basic Communication

Parameter	Present in SFC			Declaration	Data Type	Contents, Description
REQ	72	73	74	INPUT	BOOL	Initiate job with REQ = "1"
CONT	72	73	-	INPUT	BOOL	CONT = "1": Connection remains intact after job terminates
IOID	72	73	74	INPUT	BYTE	B#16#54 = Input area, B#16#55 = Output area
LADDR	72	73	74	INPUT	WORD	Module start address
VAR_ADDR	72	73	-	INPUT	ANY	Data area in partner CPU
SD	-	73	-	INPUT	ANY	Data area in own CPU which contains the Send data
RET_VAL	72	73	74	RETURN	INT	Error information
BUSY	72	73	74	OUTPUT	BOOL	Job in progress when BUSY = "1"
RD	72	-	-	OUTPUT	ANY	Data area in own CPU which will take the Receive data

BUS subnetwork or you create one either in the SIMATIC Manager (select the *Project* object and then INSERT → SUBNETWORK → PROFIBUS) or in the Network Configuration (see Chapter 2.4, "Configuring the Network").

Example: you have configured distributed I/O with a CPU 315-2DP as master. You use another CPU 315-2DP as an "intelligent" slave. You can now use station-internal S7 basic communication from both controllers to read and write data.

20.6.2 System Functions for Station-Internal S7 Basic Communication

The following system functions handle data transfers between two CPUs in the same station:

- ▷ SFC 72 I_GET
  Read data
- ▷ SFC 73 I_PUT
  Write data
- ▷ SFC 74 I_ABORT
  Disconnect

The parameters for these SFCs are listed in Table 20.15.

SFC 72 I_GET
Read data

A job is initiated with REQ = "1" and BUSY = "0" ("first call"). While the job is in progress, BUSY is set to "1". Changes to the REQ parameter no longer have any effect. When the job is completed, BUSY is reset to "0". If REQ is still "1", the job is immediately restarted.

When the read procedure has been initiated, the operating system in the partner CPU assembles and sends the requested data. An SFC call transfers the Receive data to the destination area. RET_VAL then shows the number of bytes transferred.

If CONT is = "0", the communication link is broken. If CONT is = "1", the link is maintained. The data are also read when the communication partner is in STOP mode.

The RD and VAR_ADDR parameters describe the area from which the data to be transferred are to be read or to which the receive data are to be written. Actual parameters may be addresses, variables or data areas addressed with an ANY pointer. The Send and Receive data are not checked for identical data types.

SFC 73 I_PUT
Write data

A job is initiated with REQ = "1" and BUSY = "0" ("first call"). While the job is in progress, BUSY is set to "1". Changes to the REQ parameter no longer have any effect. When the job is completed, BUSY is reset to "0". If REQ is still "1", the job is immediately restarted.

When the write procedure has been initiated, the operating system transfers all data from the source area to an internal buffer on the first call, and sends them to the partner in the link. There, the receiver writes the data into data area VAR_ADDR. BUSY is then set to "0". The data are also written when the receiving partner is at STOP.

The SD and VAR_ADDR parameters describe the area from which the data to be transferred are to be read or to which the receive data are to be written. Actual parameters may be addresses, variables or data areas addressed with an ANY pointer. The Send and Receive data are not checked for identical data types.

SFC 74 I_ABORT
Disconnect

REQ = "1" breaks a connection to the specified communication partner. With I_ABORT, you can break only those connections established in the same station with I_GET or I_PUT.

While the job is in progress, BUSY is set to "1". Changes to the REQ parameter no longer have any effect. When the job is completed, BUSY is reset to "0". If REQ is still "1", the job is immediately restarted.

20.6.3 Station-External S7 Basic Communication

Fundamentals

With station-external S7 basic communication, you can have event-driven data exchange between SIMATIC S7 stations. The stations must be connected to each other via an MPI subnetwork. The communications functions required for this are SFCs in the operating system of the CPU. These SFCs establish the communication connections themselves, if necessary. For this reason, these station-external connections are not configured via the connection table ("Communication via non-configured connections").

Station-external S7 basic communication can execute event-driven data transfer, for example, parallel to cyclic global data communication.

Addressing the nodes, connections

These functions address nodes that are on the same MPI subnetwork. The node identification is derived from the MPI address (DEST_ID parameter).

These system functions set up the required communication links dynamically and – if specified – break them when the job has been executed. If a connection cannot be established

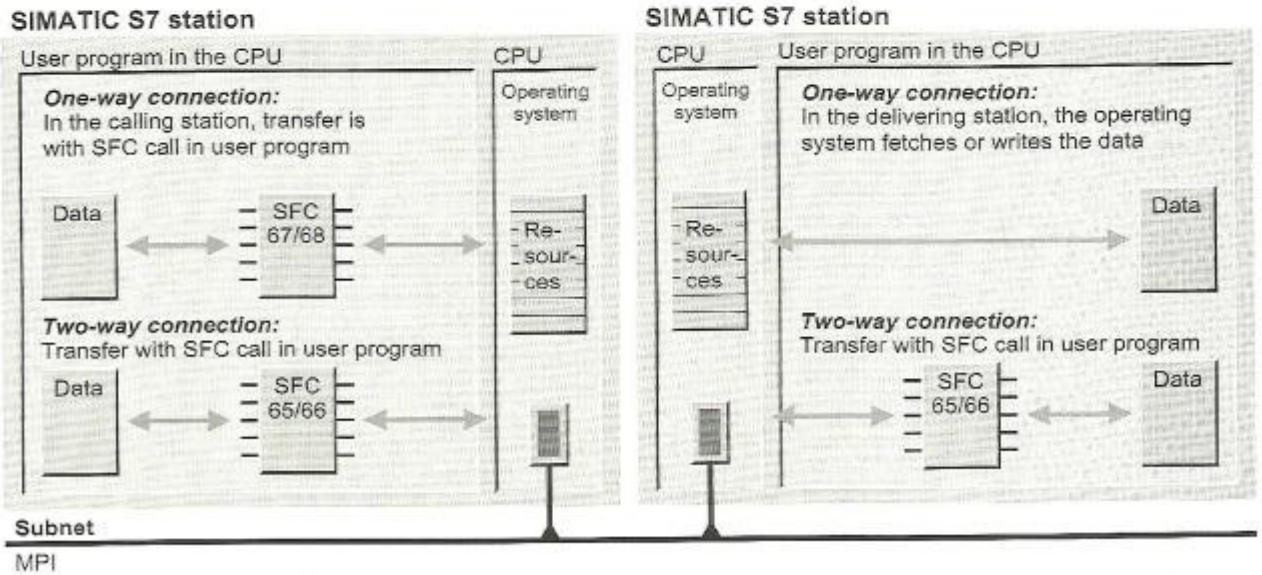


Figure 20.24 Station-External S7 Basic Communication

because of a lack of resources in either the sender or the receiver, "temporary lack of resources" is reported. The transfer must then be retried. Between two communication partners, there can be only one connection in each direction.

On a transition from RUN to STOP, all active connections (all SFCs except X_RECV) are cleared.

By modifying the block parameters at run time, you can utilize a system function for different communication links. An SFC may not interrupt itself. You may modify a program section in which one of these SFCs is used only in STOP mode; a cold or warm restart must then be executed.

User data, data consistency

These SFCs transfer a maximum of 76 bytes of user data. A CPU's operating system combines the user data into blocks consistent within themselves, without regard to the direction of transfer. The length of consistent transferred data is a CPU-specific variable.

If two CPUs exchange data via X_GET or X_PUT, the block size of the "passive" CPU is decisive to data consistency of the transferred data.

In the case of a SEND/RECEIVE connection, all data of a call are consistent.

Configuring station-external S7 basic communication

Station-external S7 basic communication is a special case in that it requires no configuring since data transfer is handled via dynamic connections. You simply use an existing MPI subnetwork or you create one.

Example: you have a divided S7-400 mounting rack with one CPU 416 in each section. In addition, you connect an S7-300 station with a CPU 314 via an MPI cable to one of the S7-400s. You configure all three CPUs in the Hardware Configuration, for example, as "networked" via an MPI subnetwork. You can now use station-external S7 basic communication from all three controllers to exchange data.

20.6.4 System Functions for Station-External S7 Basic Communication

The following system functions handle data transfers between partners in different stations:

- ▷ SFC 65 X_SEND
  Send data
- ▷ SFC 66 X_RCV
  Receive data
- ▷ SFC 67 X_GET
  Read data
- ▷ SFC 68 X_PUT
  Write data
- ▷ SFC 69 X_ABORT
  Disconnect

The parameters for these SFCs are listed in Table 20.16.

SFC 65 X_SEND
Send data

A job is initiated with REQ = "1" and BUSY = "0" ("first call"). While the job is in progress, BUSY is set to "1"; changes to the REQ parameter now no longer have any effect. When the job terminates, BUSY is set back to "0". If REQ is still "1", the job is immediately restarted.

On the first call, the operating system transfers all data from the source area to an internal buffer, then transfers the data to the partner CPU.

BUSY is "1" for the duration of the send procedure. When the partner has signaled that it has fetched the data, BUSY is set to "0" and the send job terminated.

If CONT is = "0", the connection is broken and the respective CPU resources are available to other communication links. If CONT is = "1", the connection is maintained. The REQ_ID parameter makes it possible for you to assign an ID to the Send data which you can evaluate with SFC X_RCV.

The SD parameter describes the area from which the data to be sent are to be read. Actual parameters may be addresses, variables, or data areas addressed with an ANY pointer. Send and Receive data are not checked for matching data types.

Table 20.16 SFC Parameters for Station-External S7 Basic Communication

Parameter	Present in SFC					Declaration	Data Type	Contents, Description
REQ	65	-	67	68	69	INPUT	BOOL	Job initiation with REQ = "1"
CONT	65	-	67	68	-	INPUT	BOOL	CONT = "1": Connection is maintained when job is completed
DEST_ID	65	-	67	68	69	INPUT	WORD	Partner's node identification (MPI address)
REQ_ID	65	-	-	-	-	INPUT	DWORD	Job identification
VAR_ADDR	-	-	67	68	-	INPUT	ANY	Data area in partner CPU
SD	65	-	-	68	-	INPUT	ANY	Data area in own CPU which contains the Send data
EN_DT	-	66	-	-	-	INPUT	BOOL	If "1": Accept Receive data
RET_VAL	65	66	67	68	69	RETURN	INT	Error information
BUSY	65	-	67	68	69	OUTPUT	BOOL	Job in progress when BUSY = "1"
REQ_ID	-	66	-	-	-	OUTPUT	DWORD	Job identification
NDA	-	66	-	-	-	OUTPUT	BOOL	When "1": Data received
RD	-	66	67	-	-	OUTPUT	ANY	Data area in own CPU which will accept the Receive data

SFC 66 X_RCV
Receive data

The Receive data are placed in an internal buffer. Multiple packets can be put in a queue in the chronological order of their arrival.

Use EN_DT = "0" to check whether or not data were received; if so, NDA is "1", RET_VAL shows the number of bytes of Receive data, and REQ_ID is the same as the corresponding parameter in SFC 65 X_SEND. When EN_DT is = "1", the SFC transfers the first (oldest) packet to the destination area; NDA is then "1" and RET_VAL shows the number of bytes transferred. If EN_DT is "1" but there are no data in the internal queue, NDA is "0".

On a cold or warm restart, all data packets in the queue are rejected.

In the event of a broken connection or a restart, the oldest entry in the queue, if already "queried" with EN_DT = "0", is retained; otherwise, it is rejected like the other queue entries.

The RD parameter describes the area to which the Receive data are to be written. Actual parameters may be addresses, variables, or data areas addressed with an ANY pointer.

Send and Receive data are not checked for matching data types. When the Receive data are irrelevant, a "blank" ANY pointer (NIL pointer) as RD parameter in X_RCV is permissible.

SFC 67 X_GET
Read data

A job is initiated with REQ = "1" and BUSY = "0" ("first call"). While the job is in progress, BUSY is set to "1"; changes to the REQ parameter now no longer have any effect.

When the job terminates, BUSY is set back to "0". If REQ is still "1", the job is immediately restarted.

When the read procedure has been initiated, the operating system in the partner CPU assembles and sends the data required under VAR_ADDR. On an SFC call, the Receive data are entered in the destination area specified at the RD parameter. RET_VAL then shows the number of bytes transferred.

If CONT is "0", the communication link is broken. If CONT is "1", the connection is maintained. The data are then read even when the communication partner is in STOP mode.

The RD and VAR_ADDR parameters describe the area from which the data to be sent are to be read or to which the Receive data are to be written. Actual parameters may be addresses, variables, or data areas addressed with an ANY pointer. Send and Receive data are not checked for matching data types.

SFC 68 X_PUT
Write data

A job is initiated with REQ = "1" and BUSY = "0" ("first call"). While the job is in progress, BUSY is set to "1"; changes to the REQ parameter now no longer have any effect.

When the job terminates, BUSY is set back to "0". If REQ is still "1", the job is immediately restarted.

When the write procedure has been initiated, the operating system transfers all data from the source area specified at the SD parameter to an internal buffer on the first call, then sends the data to the partner CPU. There, the partner CPU's operating system writes the Receive data to the data area specified at the VAR_ADDR parameter. BUSY is then set to "0".

The data are written even if the communication partner is in STOP mode.

The RD and VAR_ADDR parameters describe the area from which the data to be sent are to be read or to which the Receive data are to be written. Actual parameters may be addresses, variables, or data areas addressed with an ANY pointer. Send and Receive data are not checked for matching data types.

SFC 69 X_ABORT
Disconnect

REQ = "1" breaks an existing connection to the specified communication partner. The SFC X_ABORT can be used to break only those connections established in the CPU's own station with the SFCs X_SEND, X_GET or X_PUT.

20.7 S7 Communication

20.7.1 Fundamentals

With S7 communication, you transfer larger volumes of data between SIMATIC S7 stations. The stations are connected to each other via a

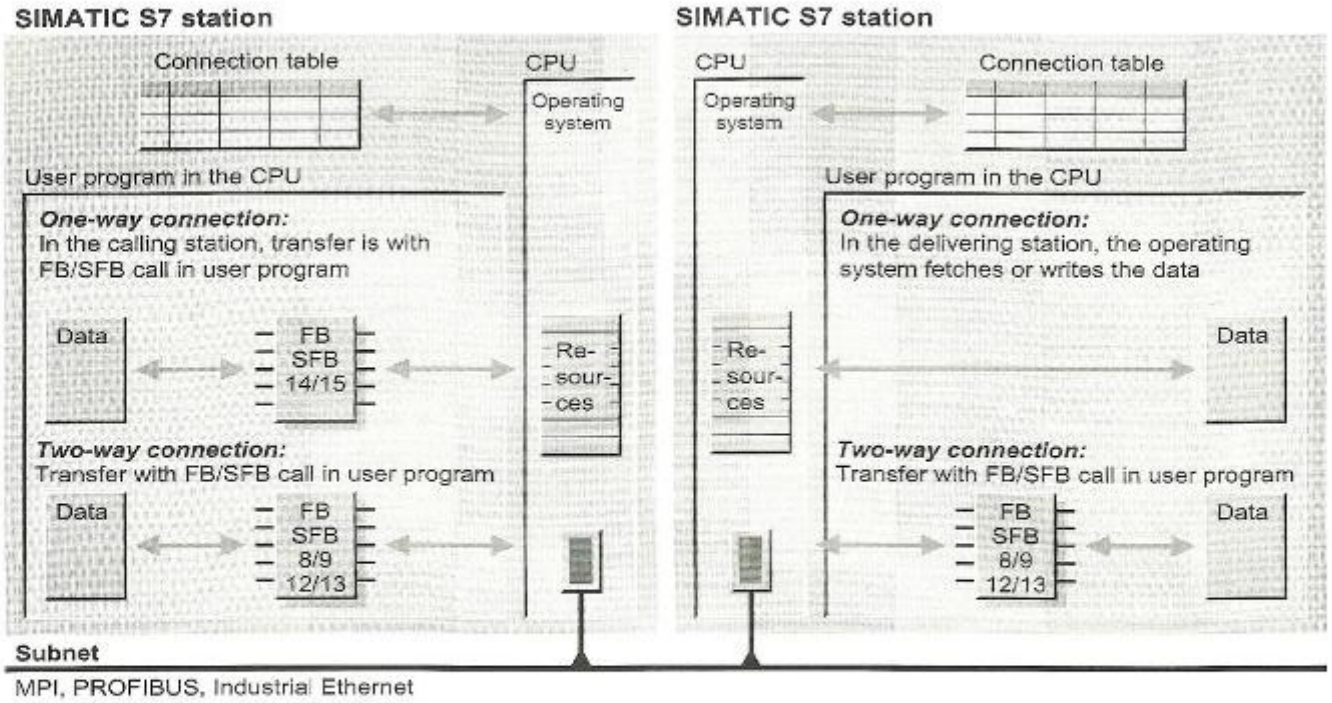


Figure 20.25 S7 Communication

subnetwork; this can be an MPI subnetwork, a PROFIBUS subnetwork or an Ethernet subnetwork. The communications connections are static; they are configured in the connection table ("Communication via configured connections").

With the S7-400, the communications functions are system function blocks SFBs integrated in the operating system of the CPUs. The associated instance data block is located in the user memory. If you want to use S7 communication, copy the interface description of the SFBs from the *Standard Library* under *System Function Blocks* to the *Blocks* container, generate an instance data block for each call and call the SFB with the associated instance data block. With incremental input, you can also select the SFB from the program element catalog and have the instance data block generated automatically.

With the S7-300, the communication functions are standard function blocks FB which you can find in the *Standard Library* under *Communication Blocks*. Copy the function blocks you wish to use into the container *Blocks* and then use them just like "normal" function blocks.

Configuring S7 communication

The prerequisite for die S7 communication is a configured connection table in which the communication links are defined.

A communication link is specified by a connection ID for each communication partner. STEP 7 assigns the connection IDs when it compiles the connection table. Use the "local ID" to initialize the FB or SFB in the module from which the connection is viewed, and the "remote ID" to initialize the FB or SFB in the partner module.

The same logical connection can be used for different Send/Receive requests. To distinguish between them, you must add a job ID to the connection ID in order to define the relationship between the Send block and Receive block.

Initialization

S7 communication must be initialized at restart so that the connection to the communication partner can be established. Initialization takes place in the CPU that receives the attribute "Active connection buildup = Yes" in the connection table. You call the communication blocks used in cyclic operation in a restart OB and initialize the parameters (provided they are available) as follows:

- REQ = FALSE
- ID = local connection ID from the connection table (data type WORD W#16#xxxx)
- R_ID = job ID which you can design for a "pair of blocks" (data type DWORD DW#16#xxxx xxxx)
- PI_NAME = variable with the contents "P_PROGRAM" in ASCII coding (e.g. ARRAY[1..9] OF CHAR).

The blocks must continue to be called in a program loop until the DONE parameter has signal state "1". The parameters ERROR and STATUS give information concerning the errors that have occurred and the job status.

You do not need to switch the data areas at restart (concerns the parameters ADDR_x, RD_x and SD_x). Exception in S7-400 systems: with SFB 8 USEND, SFB 9 URCV, SFB 14 GET and SFB 15 PUT, the communication buffers for guaranteeing consistency are created when called for the first time, and these define the maximum data quantity per transfer for all further calls.

Cyclic operation

In cyclic operation, you call the communication blocks absolutely and you control data transfer via the parameters REQ and EN_R. You must evaluate the results in the NDR, DONE, ERROR and STATUS parameters immediately following each processing of a communication block since they are only valid up to the next call.

With the S7-300, parameters of data type ANY (SD_1, RD_1, ADDR_1) can only be supplied with flags and data as the address.

20.7.2 Two-Way Data Exchange

For two-way data exchange, you require one SEND block and one RECEIVE block each at the ends of a connection. Both blocks carry the

Table 20.17 FB/SFB Parameters for Sending and Receiving Data

Parameter	Present in FB/SFB				Declaration	Data Type	Contents, Description
REQ	8	-	12	-	INPUT	BOOL	Start data exchange
EN_R	-	9	-	13	INPUT	BOOL	Receive ready
R	-	-	12	-	INPUT	BOOL	Abort data exchange
ID	8	9	12	13	INPUT	WORD	Connection ID
R_ID	8	9	12	13	INPUT	DWORD	Job ID
DONE	8	-	12	-	OUTPUT	BOOL	Job terminated
NDR	-	9	-	13	OUTPUT	BOOL	New data fetched
ERROR	8	9	12	13	OUTPUT	BOOL	Error occurred
STATUS	8	9	12	13	OUTPUT	WORD	Job status
SD_1	8	-	12	-	IN_OUT	ANY	First Send area
SD_2 [1)]	8	-	-	-	IN_OUT	ANY	Second Send area
SD_3 [1)]	8	-	-	-	IN_OUT	ANY	Third Send area
SD_4 [1)]	8	-	-	-	IN_OUT	ANY	Fourth Send area
RD_1	-	9	-	13	IN_OUT	ANY	First Receive area
RD_2 [1)]	-	9	-	-	IN_OUT	ANY	Second Receive area
RD_3 [1)]	-	9	-	-	IN_OUT	ANY	Third Receive area
RD_4 [1)]	-	9	-	-	IN_OUT	ANY	Fourth Receive area
LEN	-	-	12	13	IN_OUT	WORD	Length of data block in bytes

[1)] Not with FB 8 or FB 9

connection IDs that are located in the connection table in the same line. You can also use several "block pairs" which are then distinguished from each other by the job ID.

The following SFBs are available for two-way data interchange:

- ▷ FB/SFB 8 USEND
  Uncoordinated sending of a data packet of CPU-specific length
- ▷ FB/SFB 9 URCV
  Uncoordinated receiving of a data packet of CPU-specific length
- ▷ FB/SFB 12 BSEND
  Sending of a data block of up to 32 or 64 KB in length
- ▷ FB/SFB 13 BRCV
  Receiving of a data block of up to 32 or 64 KB in length

FB/SFB 8 and FB/SFB 9 or FB/SFB 12 and FB/SFB 13 must always be used as a pair.

The parameters for these SFBs are listed in Table 20.17.

FB 8 USEND and FB 9 URCV
SFB 8 USEND and SFB 9 URCV
Uncoordinated sending and receiving

The SD_x and RD_x parameters are used to specify the variable or the area you want to transfer. Send area SD_x must correspond to the respective Receive area RD_x. Use the parameters without gaps, beginning with 1. No values need be specified for unneeded parameters (like an FB, not all SFB parameters need be assigned values).

A positive edge at the REQ (request) parameter starts the data exchange, a positive edge at the R (reset) parameter aborts it. A "1" in the EN_R (enable receive) parameter signals that the partner is ready to receive data, "0" can be used to abort a current job.

When the NDR parameter has assumed the value "1" following the data transfer, call the block again – this time with EN_R = "0" – to prevent the receive area from being overwritten by new data during the data evaluation.

Initialize the ID parameter with the connection ID, which STEP 7 enters in the connection table for both the local and the partner (the two IDs may differ). R_ID allows you to choose a specifiable but unique job ID which must be identical for the Send and Receive block. This allows several pairs of Send and Receive blocks to share a single logical connection (as each has a unique ID).

With the S7-400, the system function blocks accept the actual values of the ID and R_ID parameters into their instance data block on the first call. The first call establishes the communication relationship (for this instance) until the next warm or cold start. With the S7-300, you can change the assignment of the ID and R_ID parameters following each completed job.

With signal state "1" in the DONE or NDR parameter, the block signals that the job terminated without error. An error, if any, is flagged in the ERROR parameter. A value other than zero in the STATUS parameter indicates either a warning (ERROR = "0") or an error (ERROR = "1").

FB 12 BSEND and FB 13 BRCV
SFB 12 BSEND and SFB 13 BRCV
Block-oriented sending and receiving

Enter a pointer to the first byte of the data area in parameter SD_x or RD_x (the length of this actual parameter determines the maximum size of the communication buffer when called for the first time, it is not evaluated with the further calls); the number of bytes of Send or Receive data is in the LEN parameter.

Up to 64 KB (32 KB with non-integrated interface) may be transferred; the data are transferred in blocks (sometimes called frames), and the transfer itself is asynchronous to the user program scan. The LEN parameter is updated following each received block.

A positive edge at the REQ (request) parameter starts the data exchange, a positive edge at the R (reset) parameter aborts it. A "1" in the EN_R (enable receive) parameter signals that the partner is ready to receive data, "0" can be used to abort a current job.

When the NDR parameter has assumed the value "1" following the data transfer, call the block again – this time with EN_R = "0" – to prevent the receive area from being overwritten by new data during the data evaluation.

Initialize the ID parameter with the connection ID, which STEP 7 enters in the connection table for both the local and the partner (the two IDs may differ). R_ID allows you to choose a specifiable but unique job ID which must be identical for the Send and Receive block. This allows several pairs of Send and Receive blocks to share a single logical connection (as each has a unique ID).

With the S7-400, the system function blocks accept the actual values of the ID and R_ID parameters into their instance data block on the first call. The first call establishes the communication relationship (for this instance) until the next warm or cold start. With the S7-300, you can change the assignment of the ID and R_ID parameters following each completed job.

With signal state "1" in the DONE or NDR parameter, the block signals that the job terminated without error. An error, if any, is flagged in the ERROR parameter. A value other than zero in the STATUS parameter indicates either a warning (ERROR = "0") or an error (ERROR = "1").

20.7.3 One-Way Data Exchange

In one-way data exchange, the communication block call is located in only one CPU. In the partner CPU, the operating system handles the necessary communication functions.

The following blocks are available for one-way data interchange

- FB/SFB 14 GET
  Read data from a partner CPU
- FB/SFB 15 PUT
  Write data to a partner CPU

Table 20.18 lists the parameters for these blocks.

The operating system in the partner CPU collects the data read with FB/SFB 14; the operating system in the partner CPU distributes the data written with FB/SFB 15. A Send or Receive (user) program in the partner CPU is not required. The partner CPU can provide the

Table 20.18 /FB/SFB Parameters for Reading and Writing Data

Parameter	Present in FB/SFB		Declaration	Data Type	Contents, Description
REQ	14	15	INPUT	BOOL	Start data exchange
ID	14	15	INPUT	WORD	Connection ID
NDR	14	-	OUTPUT	BOOL	New data fetched
DONE	-	15	OUTPUT	BOOL	Job terminated
ERROR	14	15	OUTPUT	BOOL	Error occurred
STATUS	14	15	OUTPUT	WORD	Job status
ADDR_1	14	15	IN_OUT	ANY	First data area in partner CPU
ADDR_2 [1)]	14	15	IN_OUT	ANY	Second data area in partner CPU
ADDR_3 [1)]	14	15	IN_OUT	ANY	Third data area in partner CPU
ADDR_4 [1)]	14	15	IN_OUT	ANY	Fourth data area in partner CPU
RD_1	14	-	IN_OUT	ANY	First Receive area
RD_2 [1)]	14	-	IN_OUT	ANY	Second Receive area
RD_3 [1)]	14	-	IN_OUT	ANY	Third Receive area
RD_4 [1)]	14	-	IN_OUT	ANY	Fourth Receive area
SD_1	-	15	IN_OUT	ANY	First Send area
SD_2 [1)]	-	15	IN_OUT	ANY	Second Send area
SD_3 [1)]	-	15	IN_OUT	ANY	Third Send area
SD_4 [1)]	-	15	IN_OUT	ANY	Fourth Send area

[1)] Not with FB 14 or FB 15

required communications services both at RUN and STOP. The size of the transferred consistent data blocks depends on the (server) CPU used.

A positive edge at parameter REQ (request) starts the data interchange. Set the ID parameter to the connection ID entered by STEP 7 in the connection table.

With a "1" in the DONE or NDR parameter, the block signals that the job terminated without error. An error, if any, is flagged with a "1" in the ERROR parameter. A value other than zero in the STATUS parameter is indicative of either a warning (ERROR = "0") or an error (ERROR = "1"). You must evaluate the DONE, NDR, ERROR and STATUS parameters after *every* block call.

Use the ADDR_n parameter to specify the variable or the area in the partner CPU from which you want to fetch or to which you want to send the data. The areas in ADDR_n must coincide with the areas specified in SD_n or RD_n. Use the parameters without gaps, beginning with 1. Unneeded parameters need not be specified (as in an FB, an SFB does not have to have values for all parameters).

20.7.4 Transferring Print Data

SFB 16 PRINT allows you to transfer a format description and data to a printer via a CP 441 communications processor. Table 20.19 lists the parameters for this SFB.

A positive edge at the REQ parameter starts the data exchange with the printer specified by the ID and PRN_NR parameters. The block signals an error-free transfer by setting DONE to "1". An error, if any, is flagged by a "1" in the ERROR parameter. A value other than zero in the STATUS parameter is indicative of either a warning (ERROR = "0") or an error (ERROR = "1"). You must evaluate the DONE, ERROR and STATUS parameters after *every* block call.

Enter the characters to be printed in STRING format in the FORMAT parameter. You can

Table 20.19 Parameters for SFB 16 PRINT

Parameter	Declaration	Data Type	Contents, Description
REQ	INPUT	BOOL	Start data exchange
ID	INPUT	WORD	Connection ID
DONE	OUTPUT	BOOL	Job terminated
ERROR	OUTPUT	BOOL	Error occurred
STATUS	OUTPUT	WORD	Job status
PRN_NR	IN_OUT	BYTE	Printer number
FORMAT	IN_OUT	STRING	Format description
SD_1	IN_OUT	ANY	First variable
SD_2	IN_OUT	ANY	Second variable
SD_3	IN_OUT	ANY	Third variable
SD_4	IN_OUT	ANY	Fourth variable

integrate as many as four format descriptions for variables in this string, defined in parameters SD_1 to SD_4. Use the parameters without gaps, beginning with 1; do not specify values for unneeded parameters. You can transfer up to 420 bytes (the sum of FORMAT and all variables) per print request.

20.7.5 Control Functions

The following SFBs are available for controlling the communication partner:

- SFB 19 START
  Execute a cold or warm restart in the partner controller
- SFB 20 STOP
  Switch the partner controller to STOP
- SFB 21 RESUME
  Execute a hot restart in the partner controller

These SFBs are for one-way data exchange; no user program is required in the partner device for this purpose. The parameters for them are listed in Table 20.20.

A positive edge at the REQ parameter starts the data exchange. Enter as ID parameter the connection ID which STEP 7 entered in the connection table.

With a "1" in the DONE parameter, the block signals that the job terminated without error. An error, if any, is flagged by a "1" in the ERROR parameter. A value other than zero in the STATUS parameter is indicative of either a warning (ERROR = "0") or an error (ERROR = "1"). You must evaluate the DONE, ERROR and STATUS parameters after *every* block call.

Specify as PI_NAME an array variable with the contents "P_PROGRAM" (ARRAY [1..9] OF CHAR). If you do not assign the ARG parameter, a warm start is triggered in the partner controller; if "C" is assigned to ARG, a cold start is triggered if permissible in the partner controller. The IO_STATE parameter is currently irrelevant, and need not be assigned a value.

SFB 19 START executes a cold or warm restart of the partner CPU. Prerequisite is that the partner CPU is at STOP and that the mode selector is positioned to either RUN or RUN-P.

SFB 20 STOP sets the partner CPU to STOP. Prerequisite for error-free execution of this job request is that the partner CPU is not at STOP when the request is submitted.

SFB 21 RESUME executes a hot restart of the partner CPU. Prerequisite is that the partner CPU is at STOP, that the mode selector is set to either RUN or RUN-P, and that a hot restart is permissible at this time.

20.7.6 Monitoring Functions

The following system blocks are available for monitoring functions

- SFB 22 STATUS
  Check partner status
- SFB 23 USTATUS
  Receive partner status

Table 20.20 SFB Parameters for Partner Controller

Parameter	Present in SFB			Declaration	Data Type	Contents, Description
REQ	19	20	21	INPUT	BOOL	Start data exchange
ID	19	20	21	INPUT	WORD	Connection ID
DONE	19	20	21	OUTPUT	BOOL	Job terminated
ERROR	19	20	21	OUTPUT	BOOL	Error occurred
STATUS	19	20	21	OUTPUT	WORD	Job status
PI_NAME	19	20	21	IN_OUT	ANY	Program name (P_PROGRAM)
ARG	19	-	21	IN_OUT	ANY	With "C" as the value, a cold start is triggered in the partner controller, if permissible
IO_STATE	19	20	21	IN_OUT	BYTE	Irrelevant

Table 20.21 SFB Parameters for Querying Status

Parameter	Present in SFB		Declaration	Data Type	Contents, Description
REQ	22	-	INPUT	BOOL	Start data exchange
EN_R	-	23	INPUT	BOOL	Ready to receive
ID	22	23	INPUT	WORD	Connection ID
NDR	22	23	OUTPUT	BOOL	New data fetched
ERROR	22	23	OUTPUT	BOOL	Error occurred
STATUS	22	23	OUTPUT	WORD	Job status
PHYS	22	23	IN_OUT	ANY	Physical status
LOG	22	23	IN_OUT	ANY	Logical status
LOCAL	22	23	IN_OUT	ANY	Status of an S7 CPU as partner

- ▷ SFC 62 CONTROL
  Check status of an SFB instance
- ▷ FC 62 C_CNTRL
  Check status of a connection
- ▷ SFC 87 C_DIAG
  Determine connection status

The parameters of these blocks are described in Tables 20.21, 20.22 and 20.23.

The following applies for these system blocks: an error is indicated with "1" at the ERROR parameter. If the STATUS parameter has a value not equal to zero, this indicates either a warning (ERROR = "0") or an error (ERROR = "1").

SFB 22 STATUS
Check the status of the partner device

SFB 22 STATUS fetches the status of the partner CPU and displays it in the PHYS (physical status), LOG (logical status) and LOCAL (operating status if the partner is an S7 CPU) parameters.

A positive edge at the REQ (request) parameter starts the query. Enter as ID parameter the connection ID which STEP 7 entered in the connection table.

With a "1" in the NDR parameter, the block signals that the job terminated without error. You must evaluate the NDR, ERROR and STATUS parameters after *every* block call.

SFB 23 USTATUS
Receive the status of the partner device

SFB 23 USTATUS receives the status of the partner, which it sends, unrequested, in the event of a change. The device status is displayed in the PHYS, LOG and LOCAL parameters.

Table 20.22 Parameters of the Blocks FC 62 C_CNTRL and SFC 62 CONTROL

Parameter	Present in		Declaration	Data Type	Contents, Description
EN_R	FC	SFC	INPUT	BOOL	Ready to receive
I_DB	-	SFC	INPUT	BLOCK_DB	Instance data block
OFFSET	-	SFC	INPUT	WORD	Number of the local instance
ID	FC	-	INPUT	WORD	Connection ID
RET_VAL	-	SFC	RETURN	INT	Error information
RETVAL	FC	-	RETURN	INT	Error information
ERROR	FC	SFC	OUTPUT	BOOL	Error detected
STATUS	FC	SFC	OUTPUT	WORD	Status word
I_TYP	-	SFC	OUTPUT	BYTE	Block type identifier
I_STATE	-	SFC	OUTPUT	BYTE	Actual status identifier
I_CONN	-	SFC	OUTPUT	BOOL	Connection status ("1" = connection exists)
I_STATUS	-	SFC	OUTPUT	WORD	STATUS parameter for communication instance
C_CONN	FC	-	OUTPUT	BOOL	Connection status ("1" = connection exists)
C_STATUS	FC	-	OUTPUT	WORD	Connection status

A "1" in the EN_R (enable receive) parameter signals that the partner is ready to receive data. Initialize the ID parameter with the connection ID, which STEP 7 enters in the connection table.

With a "1" in the NDR parameter, the block signals that the job terminated without error. You must evaluate the NDR, ERROR and STATUS parameters after *every* block call.

SFC 62 CONTROL
Check the status of a communication instance

In S7-400 systems, SFC 62 CONTROL determines the status of a communication instance and the associated connection in the local controller. Enter the SFB's instance data block in the I_DB parameter. If the SFB is called as local instance, specify the number of the local instance in the OFFSET parameter (zero when no local instance, 1 for the first local instance, 2 for the second, and so on).

With signal status "1" at the EN_R parameter (enable to receive), the status of the communication instance is displayed which is specified at the I_DB parameter. You must evaluate the NDR, ERROR and STATUS parameters after *every* block call.

The parameters I_TYP, I_STATE, I_CONN and I_STATUS provide information concerning the status of the local communication instance.

FC 62 C_CNTRL
Check the status of a connection

In S7-300 systems, FC 62 C_CNTRL determines the status of a connection in the local device. Enter the connection ID in the ID parameter which STEP 7 defines in the connection table for the local device.

With signal status "1" at the EN_R parameter (enable to receive), the actual connection status is displayed. The ERROR and STATUS parameters must be evaluated following each block call.

The C_CONN and C_STATUS parameters provide information on the actual connection status.

SFC 87 C_DIAG
Determine the connection status

The system function SFC 87 C_DIAG determines the actual status of connections with a fixed configuration, i.e. all S7 connections and all fault-tolerant S7 connections. With each call, the SFC 87 C_DIAG reads the connection data from the operating system and enters them into the user memory for evaluation. The SFC then acknowledges that reading has taken place in the operating system so that a change in status since the last read operation can be recorded. If you wish to monitor the connections permanently, call the SFC at regular intervals, e.g. every 10 seconds in a time interrupt organization block.

SFC 87 C_DIAG is a system function which operates in asynchronous mode. You trigger a job with signal status "1" and the REQ parameter. If the job cannot be executed immediately, the SFC returns the signal status "0" on the BUSY parameter, otherwise BUSY = "1" means that the job is in progress.

Table 20.23 shows the parameters of SFC 87 C_DIAG.

SFC 87 C_DIAG can work in various operating modes which you can set using the MODE parameter:

- MODE = B#16#00
  The SFC acknowledges reading out, without copying the connection data.
- MODE = B#16#01
  The SFC copies the connection data and acknowledges reading out.

```
DATA_BLOCK con_data          //Data block with the connection data
...
STRUCT
...
con_req          : BOOL;     //Trigger job
con_busy         : BOOL;     //Job running
con_error        : INT;      //Error information of SFC
con_index        : INT;      //Number of field elements read by SFC
con_status       : ARRAY [1..12] OF STRUCT
      CON_ID     : WORD;     //Connection ID
      STAT_CON   : BYTE;     //Connection status
      PROD_CON   : BYTE;     //Partial connection number of productive connection
      STBY_CON   : BYTE;     //Partial connection number of standby connection
      DIS_PCON   : BOOL;     //Change in status in fault tolerance
      DIS_CON    : BOOL;     //Change in status of conn. (without fault tolerance)
      RES0       : BYTE;     //Reserved
      RES1       : BYTE;     //Reserved
      END_STRUCT;
...
END_STRUCT
END_DATA_BLOCK
```

You can use your own expressions for the names of variables and components.
The call could then be as follows, for example:

```
CALL C_DIAG (
      REQ       := con_data.con_req,
      MODE      := B#16#02,
      RET_VAL   := con_data.con_error,
      BUSY      := con_data.con_busy,
      N_CON     := con_data.con_index,
      CON_ARR   := con_data.con_status);
```

Figure 20.26 Programming Example of the SFC 87 C_DIAG

Table 20.23 Parameters of the SFC 87 C_DIAG

Parameter	Declaration	Data Type	Contents, Description
REQ	INPUT	BOOL	Trigger a job with signal status "1"
MODE	INPUT	BYTE	Operating mode, see text
RET_VAL	RETURN	INT	Error information
BUSY	OUTPUT	BOOL	Job still in progress when "1"
N_CON	OUTPUT	INT	Index of the last structure
CON_ARR	OUTPUT	ANY	Destination range for the read connection data

- MODE = B#16#02
  The SFC only copies the connection data if they have changed, and acknowledges reading out – even if no changes have occurred.
- MODE = B#16#03
  The SFC copies the connection data without acknowledging.

The SFC 87 C_DIAG transmits the actual connection data from the operating system into the destination area specified in the CON_ARR parameter. The destination area is a field of structures; a field compon+ent contains the data for one connection. The number of field elements (structures) must correspond to the number of possible connections. Figure 20.26 shows the possible structure of a corresponding field variable with the connection data.

The field with the connection data is not arranged according to the connection IDs; the individual connections can be assigned in any manner to the field elements. Field elements with invalid connections can also be positioned between field elements with valid connections. The data of a connection are consistent with one another.

20.8 IE communication

20.8.1 Basics

With "Open communication via Industrial Ethernet" (IE communication for short), you transfer data between two devices connected to the Ethernet subnet. Communication can be implemented using the protocols TCP native in accordance with RFC 793, ISO-on-TCP in accordance with RFC 1006, or UDP in accordance with RFC 768.

The communication functions are loadable function blocks (FBs) contained in the *Standard Library* of STEP 7 under *Communication Blocks*. Included are user-defined data types (UDTs) with the structure of the connection data and the address of the communication partner.

Configuring IE communication

The following are required before data can be transferred with IE communication:

- In the case of the protocols TCP native and ISO-on-TCP, a connection must be established to the communication partner ("connection-oriented protocols") or
- In the case of the protocol UDP, a connection must be established to the communication layer of the CPU operating system ("connectionless protocol"). The partner is then addressed when the relevant function block is called.

The connection is configured via a data area (not using the connection table). The necessary data structures are stored in the user-defined data type UDT 65 TCON_PAR that the function blocks use for establishing and clearing down the connection. The data contain the connection ID that states a specific connection and the associated function block calls, and the information on the protocol used.

Establishment of the connection to the partner or setting up of the communication access point is handled by the function block FB 65 TCON that you call in the main program of both partner devices. Data can be transferred in parallel in both

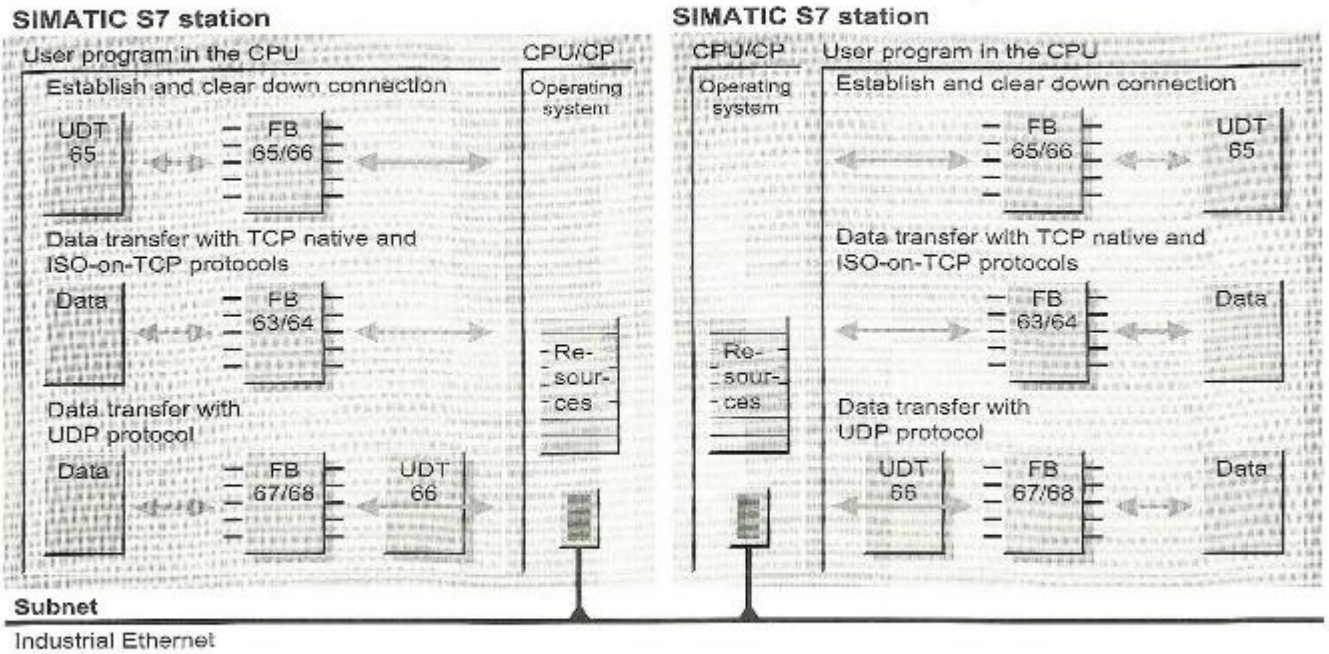


Figure 20.27 IE communication

directions over an established connection. Several connections can exist on one physical line. The function block FB 66 TDIS_CON clears the connection down again and thus releases the resources used (Figure 20.27).

With the function blocks FB 63 TSEND and FB 64 TRCV, you can transfer data with the protocols TCP native or ISO-on-TCP. Data transfer with the UDP protocol requires the function blocks FB 67 TUSEND and FB 68 TURCV. When calling these function blocks, specify the address of the partner device in a data area. The structure of this address is in the user-defined data type UDT 66 TADD_PAR.

Calling function blocks

The function blocks for IE communication work asynchronously, that is, job execution can take several program cycles under certain circumstances. You can call the communication blocks in the main program and control data transfer with the parameters REQ and EN_R. You must evaluate the results at the parameters BUSY, NDR, DONE, ERROR and STATUS immediately after each execution since they only remain valid until the next call.

20.8.2 Establishing and clearing down connections

Before data can be transferred with IE communication, a connection must be established with the partner device (in the case of TCP native and ISO-on-TCP) or with the communication layer of the operating system (in the case of UDP). The following blocks are available for this purpose:

- ▷ FB 65 TCON
  Establish connection to the communication partner or the communication layer of the operating system
- ▷ FB 66 TDISCON
  Clear down connection
- ▷ UDT 65 TCON_PAR
  Structure for the connection data

You can find the parameters of the function blocks in Table 20.24. The design of the data structure is shown in Table 20.25.

Table 20.24 Parameters for FB 65 TCON and FB 66 TDISCON

Parameters	for FB		Declaration	Data Type	Contents, Description
REQ	65	66	INPUT	BOOL	Start job (with rising edge)
ID	65	66	INPUT	WORD	Reference to communication connection
DONE	65	66	OUTPUT	BOOL	Job running ("0") or executed without errors ("1")
BUSY	65	66	OUTPUT	BOOL	Job being processed ("1") or has been completed ("0")
ERROR	65	66	OUTPUT	BOOL	Error occurred (at "1")
STATUS	65	66	OUTPUT	WORD	Job status, error information in the case of ERROR = "1"
CONNECT	65	-	IN_OUT	ANY	Pointer to the connection description

Table 20.25 Structure of the connection description UDT 65 TCON_PAR

Byte	Parameter	Data Type	Default value	Contents, Description
0 to 1	block-length	WORD	W#16#0040	Length of UDT 65 (64 bytes)
2 to 3	id	WORD	W#16#0000	Connection reference Value range: W#16#0001 to W16#0FFF
4	connection_ type	BYTE	B#16#01	Connection type B#16#01: TCP/IP native (compatibility mode) B#16#11: TCP/IP native B#16#12: ISO on TCP B#16#13: UDP
5	active_est	BOOL	"0"	Type of connection buildup: "1" active; "0" passive; with UDP: always "0"
6	local_ device_id	BYTE	B#16#02	Device identifier: ID of the communication device (see manual
7	local_tsap_ id_len	BYTE	B#16#02	Length of parameter local_tsap_id
8	rem_sub_ net_id	BYTE	B#16#00	Currently not used
9	rem_staddr_ len	BYTE	B#16#00	Length of the address of the remote connection point; not used with UDP
10	rem_tsap_ id_len	BYTE	B#16#00	Length of parameter rem_tsap_id; not used with UDP
11	next_staddr_ len	BYTE	B#16#00	Length of parameter next_staddr; not used with UDP
12 to 27	local_tsap_ id	ARRAY [1..16] OF BYTE	16(B#16#00)	Local port number or local TSAP
28 to 33	rem_subnet_ id	ARRAY [1..6] OF BYTE	6(B#16#00)	Currently not used
34 to 39	rem_staddr	ARRAY [1..6] OF BYTE	6(B#16#00)	IP address of the remote connection point; not used with UDP
40 to 55	rem_tsap_id	ARRAY [1..16] OF BYTE	16(B#16#00)	Remote port number or remote TSAP; not used with UDP
56 to 61	next_staddr	ARRAY [1..6] OF BYTE	6(B#16#00)	Rack and slot of the local CP; not used with UDP
62 to 63	spare	WORD	W#16#0000	Must be assigned W#16#0000

FB 65 TCON
Establish connection

Function block FB 65 TCON creates the prerequisites for IE communication. The parameters required for this are located in a data area that has the structure of the user-defined data type UDT 65 TCON_PAR.

When using the protocols TCP native and ISO on TCP, a connection is established to the communication partner. The station for which "Active connection setup" is entered establishes the connection. The partner station must then be designated as "passive". This designation is independent of the direction of transfer of the data.

The connection is monitored and maintained by the operating system of the CPU. In the event of a break in connection, the active partner attempts to re-establish the connection without having to call FB 65 TCON again. FB 66 TDISCON clears down the communication connection in STOP mode of the CPU or in the case of POWER OFF/ON.

If the UDP protocol is used, FB 65 TCON sets up a local communication access point that represents the connection between the user program and the communication layer of the operating system. No connection is made to the connection partner

You designate the communication connection by assigning the ID parameter. The specification must correspond to the variable *id* in the connection data. You specify the connection data with the pointer at the CONNECT parameter.

In the initial state, the parameters REQ, BUSY, DONE and ERROR have signal state "0". You start connection setup with a rising edge on the REQ parameter.

While the job is running, BUSY = "1". The job has been successfully completed when BUSY = "0", DONE = "1" and ERROR = "0". If the job contains errors, BUSY = "0", DONE = "0" and ERROR = "1". The error is then specified at the STATUS parameter. BUSY, DONE and ERROR are reset to "0" when REQ is returned to "0".

FB 66 TDISCON
Establish connection

Function block FB 66 TDISCON terminates the prerequisites for IE communication. It clears down the connection to the communication partner or deletes the communication access point.

You designate the communication connection by assigning the ID parameter. The specification must correspond to the variable *id* in the connection data.

In the initial state, the parameters REQ, BUSY, DONE and ERROR have signal state "0". You start connection cleardown with a rising edge on the REQ parameter.

While the job is running, BUSY = "1". The job has been successfully completed when BUSY = "0", DONE = "1" and ERROR = "0". If the job contains errors, BUSY = "0", DONE = "0" and ERROR = "1". The error is then specified at the STATUS parameter. BUSY, DONE and ERROR are reset to "0" when REQ is returned to "0".

UDT 65 TCON_PAR
Structure of the connection data

The user-defined data type UDT 65 TCON_PAR contains the structure of the connection data either for the communication connection to the partner device (protocols TCP native and ISO-on-TCP) or for the connection to the communication layer of the local operating system (UDP protocol).

You require a data block with this structure for each connection. For each connection, you can use your own global data block based on the UDT, or you can combine the data blocks in a shared global data block.

The assignment of the variables depends on the protocol and the devices used (see Online Help of STEP 7). UDTs with different defaults are stored in the library:

- ▷ UDT 651: for TCP active
- ▷ UDT 652: for TCP passive
- ▷ UDT 653: for ISO-on-TCP active
- ▷ UDT 654: for ISO-on-TCP passive
- ▷ UDT 655: for ISO-on-TCP active with CP
- ▷ UDT 656: for ISO-on-TCP passive with CP
- ▷ UDT 657: for open UDP local

20.8.3 Data transfer with TCP native or ISO-on-TCP

The following function blocks are available to you for data transfer with the connection-oriented protocols TCP native and ISO-on-TCP:

- FB 63 TSEND
  Send data with logic connection
- FB 64 TRCV
  Receive data with logic connection

You can find the parameters of this function blocks in Table 20.26.

You have to establish the connection to the partner station with FB 65 TCON before transferring the data (see Chapter 20.8.2, "Establishing and clearing down connections"). Data can be exchanged simultaneously in both directions over the connection with FB 63 TSEND and FB 64 TRCV.

FB 63 TSEND
Send data with logic connection

The function block FB 63 TSEND sends data with the protocols TCP native or ISO on TCP via an existing communication connection.

You can designate the communication connection by assigning the ID parameter. The specification must agree with the variable *id* in the connection data. You can specify the send mailbox with the pointer at the DATA parameter.

In the initial state, the parameters REQ, BUSY, DONE and ERROR have signal state "0". Start the data transfer with a rising edge on the REQ parameter. On the initial call with "1", the data is fetched from the area specified with the DATA parameter. The number of bytes specified at the LEN parameter is sent. Their maximum number depends on the connection type:

Connection type	Number of bytes
B#16#01	1 to 1460
B#16#11	1 to 8192
B#16#12	1 to 1452 (with CP) 1 to 8192 (without CP)

While the job is running, BUSY = "1". The job has been successfully completed when BUSY = "0", DONE = "1" and ERROR = "0". If the job contains errors, BUSY = "0", DONE = "0" and ERROR = "1". The error is then specified at the STATUS parameter. BUSY, DONE and ERROR are reset to "0" when REQ is returned to "0".

The data in the send area can then be modified again when either DONE or ERROR has signal state "1".

Table 20.26 Parameters for FB 63 TSEND and FB 64 TRCV

Parameter	with FB		Declaration	Data type	Contents, Description
REQ	63	-	INPUT	BOOL	Start data transfer (with rising edge)
EN_R	-	64	INPUT	BOOL	FB ready to receive (with "1")
ID	63	64	INPUT	WORD	Reference to communication connection
LEN	63	64	INPUT	INT	Number of bytes to be sent or received
DONE	63	-	OUTPUT	BOOL	Job running ("0") or executed without errors ("1")
NDR	-	64	OUTPUT	BOOL	Job running ("0") or completed ("1")
BUSY	63	-	OUTPUT	BOOL	Job being processed ("1") or has been completed ("0")
ERROR	63	64	OUTPUT	BOOL	Error occurred (at "1")
STATUS	63	64	OUTPUT	WORD	Job status, error information in the case of ERROR = "1"
RCVD_LEN	-	64	OUTPUT	INT	Number of bytes actually received
DATA	63	64	IN_OUT	ANY	Send or receive mailbox

FB 64 TRCV
Receive data with logic connection

The function block FB 64 TRCV receives data with the protocols TCP native or ISO on TCP via an existing communication connection.

Designate the communication connection by assigning the ID parameter. The specification must agree with the variable *id* in the connection data. Specify the receive mailbox with the pointer at the DATA parameter.

If the *LEN parameter* has 0, the length specified in the DATA parameter is used. After a data block has been received, the number of bytes received is made available on the RCVD_LEN parameter, and NDR is set to signal state "1".

With the *protocol TCP native*, neither the length of the message frame nor its start or end are transferred. So that the number of bytes sent is correctly received, the LEN parameter at the receive mailbox must be assigned the same value as the LEN parameter at the send mailbox.

If a larger value has been selected for the LEN at the receive mailbox, part of the following message frame (from the next job) will also be received. NDR is only set to "1" when the parameterized length has been reached.

If a smaller value has been selected for LEN, NDR is set to "1" when the parameterized length is reached, and the RCVD_LEN parameter is assigned the number of received bytes. With each subsequent call, another data block is received .

Information about the length and end of a message frame is sent with the *protocol ISO-on-TCP*. If LEN at the receive mailbox is larger than at the send mailbox, the sent data are received, NDR is set to "1", and the number of received bytes is written in RCVD_LEN. If LEN is smaller, an error message is issued: ERROR = "1", STATUS = W#16#8088.

FB 64 TRCV only receives data when the EN_R parameter has signal state "1".

While the job is running, BUSY = "1". The job has been successfully completed when BUSY = "0", NDR = "1" and ERROR = "0". If the job contains errors, BUSY = "0", NDR = "0" and ERROR = "1". The error is then specified at the STATUS parameter. BUSY, NDR and ERROR are reset to "0" when EN_R is returned to "0".

The data in the receive mailbox is consistent when NDR has signal state "1".

20.8.4 Data transfer with UDP

The following blocks are available to you for data transfer with the connectionless UDP protocol:

▷ FB 67 TUSEND
Send data with UDP

Table 20.27 Parameters for FB 67 TUSEND and FB 68 TURCV

Parameters	with FB		Declaration	Data type	Contents, Description
REQ	67		INPUT	BOOL	Start data transfer (with rising edge)
EN_R		68	INPUT	BOOL	FB ready to receive (with "1")
ID	67	68	INPUT	WORD	Reference to the communication point
LEN	67	68	INPUT	INT	Number of bytes to be sent or received
DONE	67	-	OUTPUT	BOOL	Job running ("0") or executed without errors ("1")
NDR	-	68	OUTPUT	BOOL	Job running ("0") or completed ("1")
BUSY	67	-	OUTPUT	BOOL	Job being processed ("1") or has been completed ("0")
ERROR	67	68	OUTPUT	BOOL	Error occurred (at "1")
STATUS	67	68	OUTPUT	WORD	Job status, error information in the case of ERROR = "1"
RCVD_LEN	-	68	OUTPUT	INT	Number of bytes actually received
DATA	67	68	IN_OUT	ANY	Send or receive mailbox
ADDR	67	68	IN_OUT	ANY	Pointer to the address of the sender or receiver

Table 20.28 Structure of the partner address UDT 66 TADD_PAR

Byte	Parameter	Data type	Default value	Contents, Description
0 to 3	rem_ip_addr	ARRAY [1..4] OF BYTE	4(B#16#00)	IP address of the partner
4 to 5	rem_port_nr	ARRAY [1..2] OF BYTE	2(B#16#00)	Port number of the partner
6 to 7	spare	ARRAY [1..2] OF BYTE	2(B#16#00)	Must be assigned 0000

- FB 68 TURCV
  Receive data with UDP
- UDT 66 TADD_PAR
  Data structure for the partner address

Table 20.27 shows the parameters of the function blocks, and Table 20.28 shows the structure of the UDT.

You have to establish a connection to the communication layer of the operating system with FB 65 TCON before transferring the data (see Chapter 20.8.2, "Establishing and clearing down connections"). The address of the communication partner is located in a data area that has the structure of UDT 66 TADD_PAR.

FB 67 TUSEND
Send data with UDP

The function block FB 67 TUSEND sends data with the UDP protocol.

The assignment of the ID parameter designates the connection between the user program and the communication layer of the operating system. The value must agree with the variable *id* in the connection data. Specify the send mailbox with the pointer at the DATA parameter.

The information on the communication partner is located in a data area to which the pointer at the ADDR parameter points. With each new send job, the address and thus the partner can be changed without having to redefine the communication access point with FB 65 TCON.

In the initial state, the parameters REQ, BUSY, DONE and ERROR have signal state "0". Start data transfer with a rising edge on the REQ parameter. On the initial call with "1", the data is fetched from the area specified with the DATA parameter. The number of bytes specified at the LEN parameter is sent (1 to max. 1,460).

While the job is running, BUSY = "1". The job has been successfully completed when BUSY = "0", DONE = "1" and ERROR = "0". If the job contains errors, BUSY = "0", DONE = "0" and ERROR = "1". The error is then specified at the STATUS parameter. BUSY, DONE and ERROR are reset to "0" when REQ is returned to "0".

The data in the send area can then be modified again when either DONE or ERROR has signal state "1".

FB 68 TURCV
Receive data with UDP

The function block FB 68 TURCV receives data with the UDP protocol.

The assignment of the ID parameter designates the connection between the user program and the communication layer of the operating system. The value must agree with the variable *id* in the connection data. Specify the receive mailbox with the pointer at the DATA parameter.

The information on the communication partner is located in a data area to which the pointer at the ADDR parameter points.

The number of bytes to be received is set at the LEN parameter (1 to max. 1,460). After a data block has been received, the number of bytes received is made available on the RCVD_LEN parameter, and NDR is set to signal state "1".

Data is only received when the EN_R parameter has signal state "1".

While the job is running, BUSY = "1". The job has been successfully completed when BUSY = "0", NDR = "1" and ERROR = "0". If the job contains errors, BUSY = "0", NDR = "0" and ERROR = "1". The error is then specified at the STATUS parameter. BUSY, NDR and ERROR are reset to "0" when EN_R is returned to "0".

The data in the receive area is consistent when NDR has signal state "1".

UDP (User Data Protocol)

Connection buildup is not carried out with UDP. The communication partner is specified at the ADDR parameter of the send block (IP address and port number). The receive block then supplies the IP address and the port number of the sender at the ADDR parameter.

The user-defined data type UDT 66 TADD_PAR contains the structure of the address information. The pointer at ADDR points to a data area with this structure.

With UDP, information on the length and end of a message frame is transferred. If LEN at the receive mailbox is larger, the sent data is copied to the receive mailbox, NDR is set to "1", and the number of received bytes is written in RCVD_LEN. If LEN is smaller, an error message is issued: ERROR = "1", STATUS = W#16#8088.

UDT 66 TADD_PAR
Data structure of the partner address

UDT 66 contains the structure of the partner address in the case of transfer with the UDP protocol. The ADDR parameter at the function blocks FB 67 TUSEND and FB 68 TURCV points to a data area with this structure.

20.9 PtP communication with S7-300C

20.9.1 Fundamentals

When using point-to-point communication (PtP), you transmit data over a serial interface to a communications partner, e.g. a printer or a SIMATIC S5 station. With certain S7-300 compact CPUs, an RS 422/485 interface (X.27) is already integrated.

The communications connections are specified using the Hardware Configuration tool when parameterizing the CPU's interface properties. ASCII mode, the 3964(R) procedure and the RK512 computer coupling are available as transmission protocols.

The communications functions are system function blocks SFB which are integrated in the operating system of the S7-300C CPU. The instance data blocks for these SFBs are present in the user memory. The SFBs do not test the parameters. If parameters are set incorrectly, it may occur that the CPU goes to STOP. If you use the transmission protocol of the computer coupling, a synchronization data block is additionally used (once for all computer coupling SFBs in the user memory, Figure 20.28).

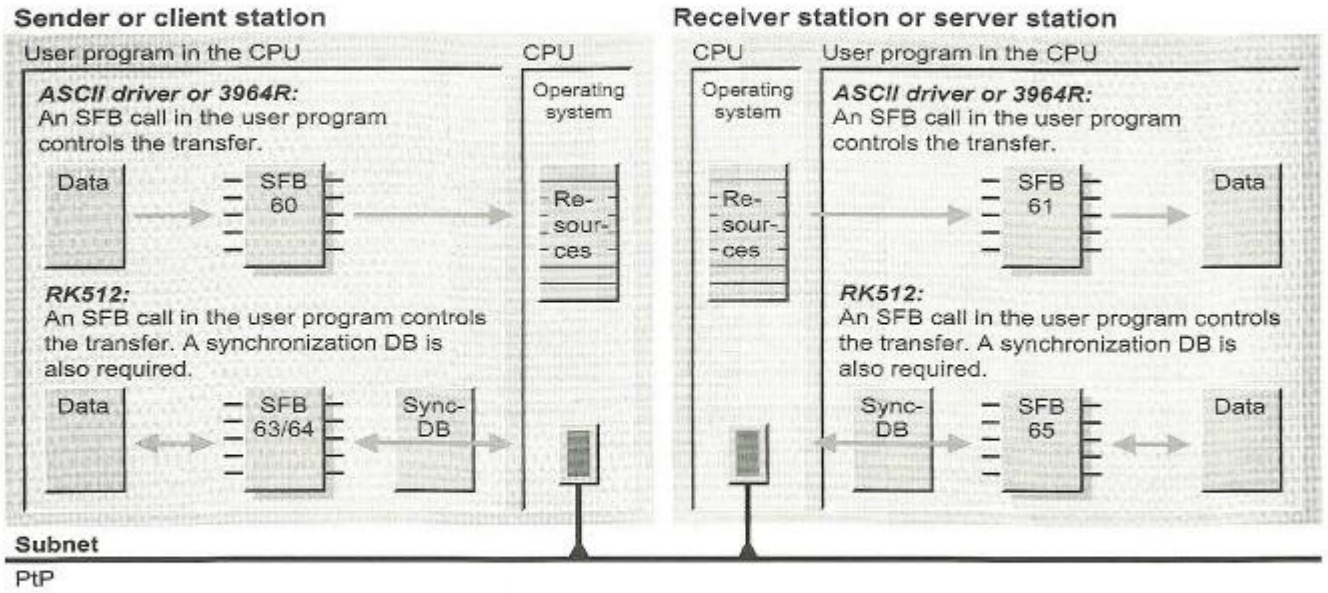


Figure 20.28 Point-to-point Connection with S7-300C

Configuring PtP communication

Use the Hardware Configuration tool to set the transmission protocol in the properties window of the point-to-point interface:

- ASCII mode
  The data are transmitted as ASCII characters. The transmission is not acknowledged. Setting of signal assignments and transmission parameters such as baud rate, parity, end-of-text character.

- 3964(R) procedure
  The data are sent to the communications partner, and positively acknowledged by this when received correctly. Setting of signal assignments and transmission parameters such as baud rate, parity, block check.

- RK512 computer coupling
  Data transmission can be coordinated by so-called communication flags. Data receipt and the fetching of data are acknowledged. Setting of signal assignments and transmission parameters such as baud rate, parity, block check.

With the 3964(R) and RK512 transmission protocols, the communications partners must have different priorities in order to define the sequence when requests to transmit are sent simultaneously.

20.9.2 ASCII driver and 3964(R) procedure

You can send and receive data via the point-to-point connection when using the ASCII driver or the 3964(R) procedure. The following system function blocks are required:

- SFB 60 SEND_PTP
  Send data with ASCII driver or 3964(R) procedure

- SFB 61 RCV_PTP
  Receive data with ASCII driver or 3964(R) procedure

- SFB 62 RES_RCVB
  Delete receive buffer with ASCII driver or 3964(R) procedure

Table 20.29 shows the parameters of these system function blocks.

SFB 60 SEND_PTP
Send data with ASCII driver or 3964R procedure

The system function block SFB 60 SEND_PTP is used to send a data area to a communications partner. Use the Hardware Configuration tool to set the transmission protocol and the transmission parameters. Specify the area of data to be transmitted using the SD_1 parameter. The length of the transmitted data area depends on the interface parameters, e.g.

Table 20.29 SFB Parameters for Sending and Receiving Data with ASCII Driver or 3964(R) Procedure

Parameter	Present in SFB			Declaration	Data Type	Contents, Description
REQ	60	-	62	INPUT	BOOL	Trigger job with signals status "1"
EN_R	-	61	-	INPUT	BOOL	Receive enable
R	60	61	62	INPUT	BOOL	With "1", the job is aborted
LADDR	60	61	62	INPUT	WORD	Submodule address of interface
DONE	60	-	62	OUTPUT	BOOL	With "1", the job is still running
NDR	-	61	-	OUTPUT	BOOL	With "1", the job has been completed without fault
ERROR	60	61	62	OUTPUT	BOOL	With "1", a fault has occurred
STATUS	60	61	62	OUTPUT	WORD	Error information
SD_1	60	-	-	IN_OUT	ANY	Send mailbox
RD_1	-	61	-	IN_OUT	ANY	Receive mailbox
LEN	60	61	-	IN_OUT	INT	Number of transmitted bytes

send up to length specified by the LEN parameter, or send up to an end-of-text character.

With the ASCII driver, you can send telegrams up to a length of 1024 bytes. SFB 60 SEND_PTP transmits the data in consistent blocks of 206 bytes. You must not change the data in the send area while the transmission is running.

Sending is triggered by a rising signal edge on the REQ parameter. With signals status "1" on the DONE parameter, the SFB signals that the job has been completed successfully. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

You can abort a current send job by means of signal status "1" on the R parameter, and reset the call instance to the basic state.

SFB 61 RCV_PTP
Receive data with ASCII driver or 3964R procedure

The system function block SFB 61 RCV_PTP is used to receive a data area from a communications partner. Use the Hardware Configuration tool to set the transmission protocol and the transmission parameters. The received data are entered into the area specified by the RD_1 parameter. The number of received bytes is present in the LEN parameter.

SFB 61 RCV_PTP receives the data in consistent blocks of 206 bytes. You must not access the data in the receive area while the transmission is running.

The CPU's internal receive buffer has a size of 2048 bytes. In the parameter settings of the interface, you can also define whether you wish to use the complete length of the receive buffer for data receipt or to limit the number of received telegrams.

A signal status "1" on the EN_R parameter enables data receipt. The NDR parameter has a signal status "1" if new data have been received successfully. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

You can abort a current send job by means of signal status "1" on the R parameter, and reset the call instance to the basic state.

SFB 62 RES_RCVB
Delete receive buffer with ASCII driver or 3964R procedure

System function block SFB 62 RES_RCVB deletes the receive buffer of the point-to-point interface. A telegram received during the delete operation is exempted from this.

Deleting is triggered by the rising signal edge on the REQ parameter. With signal status "1" on the DONE parameter, the SFB signals that deleting was successful. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter. DONE, ERROR and STATUS are only set for one call at a time.

You can abort a current send job by means of signal status "1" on the R parameter, and reset the call instance to the basic state.

20.9.3 RK512 computer coupling

With the point-to-point connection, you can send and receive data with application of the RK512 computer coupling. The system function blocks required for this are:

- SFB 63 SEND_RK
  Send data with RK512 computer coupling
- SFB 64 FETCH_RK
  Fetch data with RK512 computer coupling
- SFB 65 SERVE_RK
  Receive data and make available with RK512 computer coupling

Table 20.30 shows the parameters of these system function blocks.

Specification of transmission area

SFB 63 SEND_RK sends from the area specified by SD_1, and addresses a data block area in the partner device. SFB 64 FETCH_RK can access all address areas in the partner device, and stores the fetched data in the data block area specified by the RD_1 parameter.

SFB 65 SERVE_RK can save received data in a data block, and make data available from all address areas. The permissible assignments for the parameters are listed in Table 20.31. Certain values are only meaningful in conjunction with a SIMATIC S5 station as the partner device.

Table 20.30 SFB Parameters for Sending and Receiving Data with RK512 Computer Coupling

Parameter	Present in SFB			Declaration	Data Type	Contents, Description
SYNC_DB	63	64	65	INPUT	INT	Number of the synchronization data block
REQ	63	64	-	INPUT	BOOL	Trigger job with "1"
EN_R	-	-	65	INPUT	BOOL	Enable receipt with "1"
R	63	64	65	INPUT	BOOL	Abort job with "1"
LADDR	63	64	65	INPUT	WORD	Logical basic address of interface
R_CPU	63	64	-	INPUT	INT	CPU number of partner station
R_TYPE	63	64	-	INPUT	CHAR	Type of data block in partner CPU
R_DBNO	63	64	-	INPUT	INT	Number of data block in partner CPU
R_OFFSET	63	64	-	INPUT	INT	Number of start byte in partner CPU
R_CF_BYT	63	64	-	INPUT	INT	Number of communication flag byte in partner CPU
R_CF_BIT	63	64	-	INPUT	INT	Number of communication flag bit in partner CPU
DONE	63	64	-	OUTPUT	BOOL	With "1": job completed without error
NDR	-	-	65	OUTPUT	BOOL	With "1": job completed without error
ERROR	63	64	65	OUTPUT	BOOL	With "1": job completed with error
STATUS	63	64	65	OUTPUT	WORD	Error information
L_TYPE	-	-	65	OUTPUT	CHAR	Type of data area on local CPU
L_DBNO	-	-	65	OUTPUT	INT	Number of data block in local CPU
L_OFFSET	-	-	65	OUTPUT	INT	Number of start byte in local CPU
L_CF_BYT	-	-	65	OUTPUT	INT	Number of communication flag byte in local CPU
L_CF_BIT	-	-	65	OUTPUT	INT	Number of communication flag bit in local CPU
SD_1	63	-	-	IN_OUT	ANY	Send mailbox
RD_1	-	64	-	IN_OUT	ANY	Receive mailbox
LEN	63	64	65	IN_OUT	INT	Number of data bytes

Synchronization data block

The system function blocks of the computer coupling work together with a synchronization data block in addition to the instance data block, and this synchronizes and controls the activities of all computer coupling instances. The synchronization data block is present once in the user memory. You should create it as a global data block with a minimum length of 240 bytes. Enter the number of the data block in the SYNC_DB parameter.

Coordination with communication flags

Receipt of data using the computer coupling can be coordinated by communication flags. A communication flag is a bit from the flag address area F. Use a communication flag for each transmission job, and specify its address in the R_CF_BYT and R_CF_BIT or L_CF_BYT and L_CF_BIT parameters.

If the local CPU is the client, the system function block SFB 63 SEND_RK is used to send data and SFB 64 FETCH_RK to fetch data. When sending and fetching data from the partner CPU, the address of the communication flag is also included. If this communication flag in the partner CPU has the signal status "0", the partner CPU permits importing of the data package into the user memory in the case of a send job, and reading of the data package from the user memory in the case of a fetch job. The communication flag is then set by the commu-

Table 20.31 Specification of Transmission Area

Parameter	Type:	SFB 63 Send data	SFB 64 Fetch data	SFB 65 Receive data	SFB 65 Provision of data	Meaning
R_CPU	INT	0 to 4	0 to 4	-	-	0 = single-processor mode 1..4 = number of CPU in multi-processor mode
R_TYPE	CHAR	D, X	D, X, M, E, A, T, C	-	-	D = data block DB X = extended data block DX M = flag memory area E = process image of inputs A = process image of outputs T = timer values Z = counter values
L_TYPE	CHAR	-	-	D	D, M, E, A, T, Z	
R_DBNO	INT	0 to 255	0 to 255	-		Number of data block (irrelevant with M, E, A, T and Z)
L_DBNO	INT	-	-	1 to n [1]	1 to n [1]	
R_OFFSET	INT	0 to 510	0 to 510	-	-	First byte with data blocks (must be an even address)
		-	0 to 255	-	-	First byte with M, E, A, T and Z
L_OFFSET	INT	-	-	0 to 1024	0 to 1024	Telegram length

[1] CPU-specific

nications function to indicate that the data transmission has taken place. The data can then be edited or preprocessed by the user program. If the communication flag is reset by the user program, the data transmission is enabled again. The communication flag in the partner CPU therefore permits control of data transmission.

If the local CPU is the server, the data are received by SFB 65 SERVE_RK if the client sends data, or made available if the client fetches data. On the SFB, parameterize the local communication flag (in the server) with which you then control the receipt or provision of data in the user program. The SFB indicates in the L_CF_BYT and L_CF_BIT parameters which communication flag is used for the currently executed job and has been set to "1". Following processing of the data (fetching or provision again), reset the communication flag by the program, thus enabling processing of the next transmission job.

SFB 63 SEND_RK
Send data with RK512 computer coupling

The system function block SFB 63 SEND_RK is used to send a data area to a communications partner. Use the Hardware Configuration tool to set the transmission protocol and the transmission parameters. Specify the data area to be sent in the SD_1 parameter. Enter the length of the sent data area in the LEN parameter. Note that the number of bytes must be even.

SFB 63 SEND_RK only transmits the data if the communication flag in the communications partner has the signal status "0". A telegram can be up to 1024 bytes long. The data are transmitted in consistent blocks of 128 bytes. You must not change the data in the send area while a transmission is taking place.

Sending is triggered by the rising signal edge on the REQ parameter. With signal status "1" on the DONE parameter, the SFB signals that the job was completed successfully. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

You can abort a current send job by means of signal status "1" on the R parameter, and reset the call instance to the basic state.

SFB 64 FETCH_RK
Fetch data with the RK512 computer coupling

The system function block SFB 64 FETCH_RK is used to fetch a data area from a communications partner. Use the Hardware Configuration tool to set the transmission protocol and the transmission parameters. The fetched data are entered into the area specified by the RD_1 parameter. The number of received bytes is present in the LEN parameter.

SFB 64 FETCH_RK only fetches the data if the communication flag in the communications partner has the signal status "0". A telegram can be up to 1024 bytes long. The data are transmitted in consistent blocks of 128 bytes. You must not change the data in the send area while a transmission is taking place.

You enable the fetching of data by means of signal status "1" in the EN_R parameter. Signal status "1" in the NDR parameter indicates that new data have been fetched successfully. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

You can abort a current fetch job by means of signal status "1" on the R parameter, and reset the call instance to the basic state.

SFB 65 SERVE_RK
Receive data and make available with the RK512 computer coupling

The system function block SFB 65 SERVE_RK has the server functionality for the RK512 computer coupling. It accepts a data area sent by a communications partner, and makes a data area available which is fetched by a communications partner. The received data, or data made available, are entered into the area specified by the L_TYPE, L_DBNO and L_OFFSET parameters. The number of transmitted bytes is present in the LEN parameter.

SFB 65 SERVE_RK transmits the data in consistent blocks of 128 bytes. You must not access the data in the transmission area while a transmission is taking place. Coordination of data transmission is controlled by a communication flag in the user program.

You enable processing of the job by signal status "1" in the EN_R parameter. Signal status "1" in the NDR parameter indicates that new data have been successfully received or fetched. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

You can abort a current job by means of signal status "1" on the R parameter, and reset the call instance to the basic state.

20.10 Configuration in RUN

Configuration in RUN (CiR) means that changes to the system can be made during operation. This functionality permits you to change the configuration of the distributed I/O of an S7-400 station without the CPU entering STOP or having to be set to STOP.

The possible changes include the addition of compact DP slaves, ET200M stations and PA master systems to an existing DP master system, the addition of modules to ET200M stations, and the addition of PA slaves (field devices) to existing PA master systems. All objects added during runtime can also be removed again (Figure 20.29).

Prerequisites and limitations exist in addition to the fact that all involved devices must be able to handle the CiR functionality. For example, the PROFIBUS DP master system must be a monomaster system and must not exhibit constant bus cycle times, the use of intelligent DP slaves is not permissible in the associated station components, and the module parameters must be saved on the CPU.

Components with and without CiR functionality can be mixed; however, changes are only possible on components with CiR capability.

During reconfiguration, processing is interrupted for a short period (typically 1 s, can be programmed). The time can be kept short if only a few changes are always carried out.

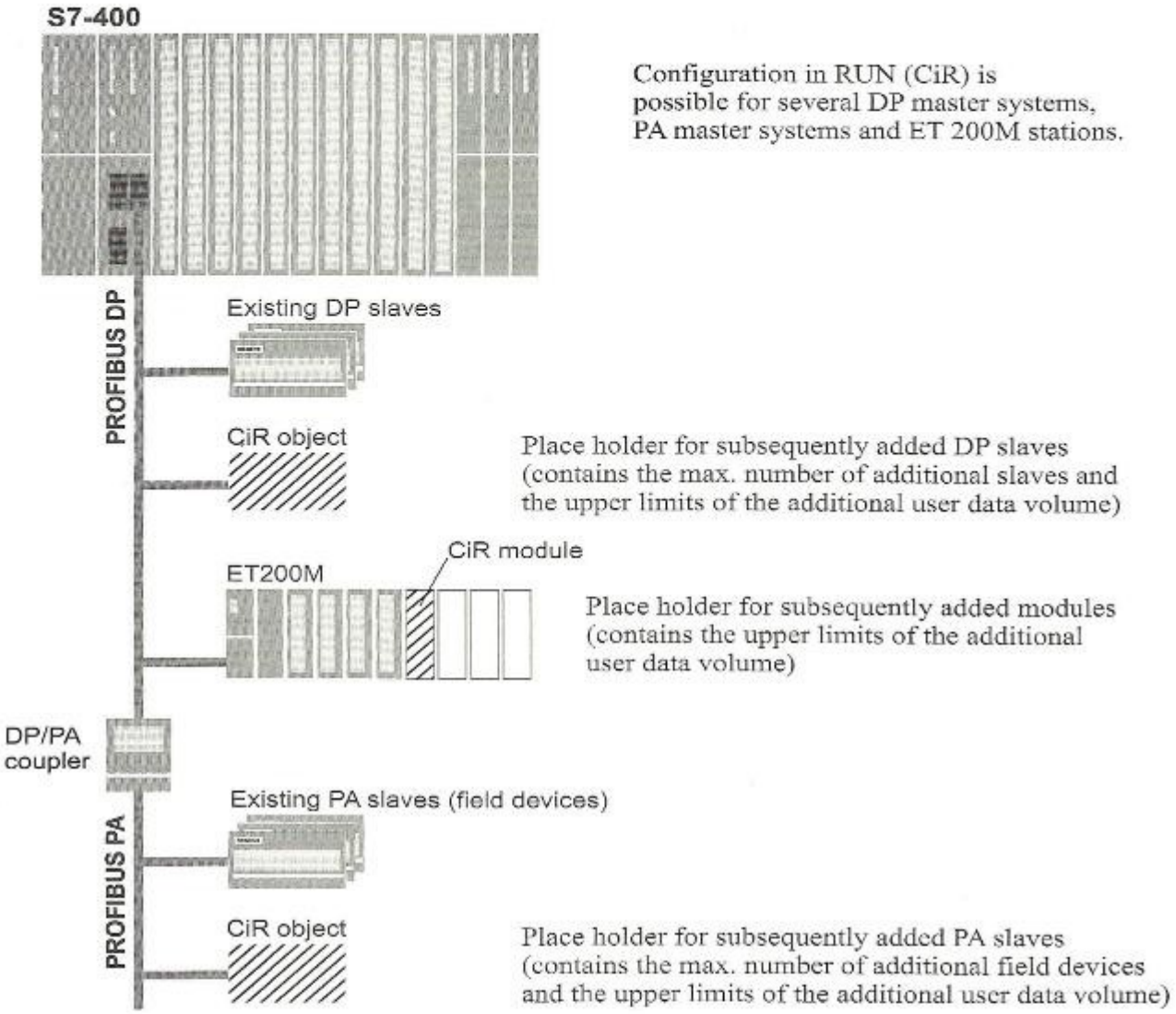


Figure 20.29 CiR Elements in the Hardware Configuration

20.10.1 Preparation of Changes in Configuration

Configure an S7-400 station (CPU with firmware release V3.1 or later) with at least one PROFIBUS DP master system using the Hardware Configuration tool. Then add the DP slaves and – if envisaged – the dummy values for later system expansion (CiR object under PROFIBUS DP in the hardware catalog). Set the later maximum configuration in the properties of this dummy value. The Hardware Configuration tool calculates 244 input bytes and 244 output bytes for each additional DP slave. In reality, far fewer user data are usually required. You can change the total of all required input and output bytes by clicking the control box "Enhanced settings".

You can also provide space in an ET200M station for later expansions. Add a station with an interface module IM 153-2BA00-0XB0 or better to the DP master system, and activate the checkbox "Module replacement during runtime" on the "Special" tab in the properties of the station. The active bus modules required to carry out the module replacements are displayed in the bottom window area (they are required for the mechanical design, but are not configured). It is recommendable to fit the ET200M station with active bus modules up to

the planned configuration since they must not be inserted or removed during runtime.

You can now insert modules into the ET200M station and – if you wish to carry out later expansions – also a dummy directly following the last configured module (the object *CiR module* under the used IM 153 in the hardware catalog). Set the required number of additional input and output bytes in the properties of the CiR module.

If you wish to extend a PA master system connected to the DP master system, use an IM 157-0AA82-0XA0 or better as the DP/PA link. A dummy is also provided for this for subsequent expansion with field devices (*CiR object* under the used IM 157 in the hardware catalog). Set the required number of additional input and output bytes in the properties of the CiR module.

With the CiR-capable DP master system marked and by using EDIT → MASTER SYSTEM → ENABLE CIR CAPABILITY, a CiR object is generated on the DP master system and on every subordinate PA master system with CiR capability. A CiR module is inserted into each ET200M station with CiR capability. All CiR objects and CiR modules are deleted again using EDIT → MASTER SYSTEM → DISABLE CIR CAPABILITY.

20.10.2 Change Configuration

You can now change the configuration within the limits specified in the CiR elements, and load again in RUN. Possible modifications are:

- Adding of compact and modular DP slaves to an existing DP master system (the added slaves must have a higher PROFIBUS address than the largest address previously used)
- Changing of assignment of partial process image with existing DP slaves
- Adding of PA slaves (field devices) to an existing PA master system
- Adding of DP/PA couplers after an IM 157 interface module
- Adding of DP/PA links including PA master system to an existing DP master system
- Adding of modules to an ET200M station
- Changing of module parameters in an ET200M station (e.g. new or modified assignment to a partial process image, activation of previously unused channels)
- Cancellation of above-mentioned modifications (starting at the highest addresses for modules and slaves)

The total of configured (real, immediately used) addresses and the addresses for future use must not be greater than the quantity framework of the DP master (is checked during configuring), but can be larger than the quantity framework of the CPU (is only checked when "converting" into specific slaves or modules).

During a CiR process, the configuration can be changed on a maximum of 4 DP master systems. Under certain conditions it is recommendable or even essential to carry out the CiR process more than once. If, for example, modules or slaves are to be replaced by others, the corresponding component must be removed first, and the replacement added in a second CiR process.

20.10.3 Load Configuration

The (initial) loading of a configuration with CiR elements or with a modified CiR configuration is carried out with the CPU in the STOP status. In order to check whether the CiR capability is also present, you should subsequently load the CiR configuration again in RUN. Testing for CiR capability using STATION → CHECK CIR COMPATIBILITY is not 100% possible offline. For example, the CiR synchronization time could be limited by the SFC 104 CiR.

To keep the CPU in the RUN status during the CiR process, you must make sure that interrupts from unknown components are ignored. A corresponding program must be present in the following organization blocks:

- Process interrupts OB 40 to OB 47
- Timing error OB 80
- Diagnostic interrupt OB 82
- Hot swapping interrupt OB 83
- Program execution fault OB 85
- Rack failure OB 86
- I/O access fault OB 122

When adding modules or slaves, you should first load the configuration and then the matching user program. When removing modules and slaves, first load the adapted user program and then the modified configuration. Adding or removing real modules or slaves must only be carried out following loading of the modified configuration (when the INTF LED on the CPU has gone off).

When reparameterizing modules you must first load a user program which no longer addresses the associated modules or no longer evaluates their interrupts. Then load the modified configuration, change the hardware if necessary, and load the user program adapted to the modification.

20.10.4 CiR Synchronization Time

Following loading of the new configuration into the CPU, the new data are checked and imported into the current configuration if correct. This importing requires a certain time, the so-called CiR synchronization time. Process execution is interrupted during this period.

The CiR synchronization time is calculated from the total of the CiR synchronization times of all DP and PA master systems involved. The synchronization time for a master system depends on the CPU used and on the real and planned I/O volumes in this master system. This time is indicated by the Hardware Configuration tool in the properties of the CiR object in the master system. The worst case is always calculated, so that the actual CiR synchronization time is shorter. The synchronization time is 100 ms if only modules are reparameterized.

The CPU compares the calculated CiR synchronization time with the permissible upper limit, whose default setting is 1000 ms. You can change this upper limit using the SFC 104 CIR. If the calculated CiR synchronization time is greater than this upper limit, the change in configuration is not carried out.

20.10.5 Effects on Program Execution

Execution of the user program is stopped during the CiR synchronization time. All process images retain their last values. The SIMATIC timers and the CPU clock continue. Any interrupts which occur are only processed at the end of the CiR synchronization time. Communication with a connected programming device is limited; only the STOP command is accepted.

Following synchronization, the CPU starts the organization block OB 80 "Timing error" with the value W#16#350A in the first word of the start information (variables OB80_EV_CLASS and OB80_FLT_ID). The required CiR synchronization time in ms is present in the variable OB80_ERROR_INFO.

If modules are to be reparameterized, the CPU then starts the organization block OB 83 "Hot swapping interrupt" with the value W#16#3367 in the first word of the start information (variables OB83_EV_CLASS and OB83_FLT_ID). The modules are subsequently reparameterized. It can occur that the associated modules do not deliver valid values in the meantime.

Following reparameterization, the CPU starts the OB 83 again, this time with W#16#3267 in the first word of the start information. Faulty reparameterization is signaled by W#16#3968. The associated modules are then considered as not available. The described procedure is executed in each master system which is affected.

20.10.6 Control CiR Process

You can use the **SFC 104 CIR** to block the CiR process in the user program, to limit it for a certain period, or to enable it. The SFC parameters are listed in Table 20.32.

MODE = B#16#00 delivers the currently valid upper limit for the CiR synchronization time. MODE = B#16#01 sets the CiR synchronization time to the default value 1000 ms and enables execution of the CiR process. MODE = B#16#02 unconditionally disables the CiR process, MODE = B#16#03 only disables it if the CiR synchronization time calculated in the CPU is greater than that specified in the FRZ_TIME parameter.

Table 20.32 Parameters of the SFC 104 CIR

Parameter	Declaration	Data Type	Contents, Description
MODE	INPUT	BYTE	Job identification B#16#00: information function B#16#01: enable CiR process B#16#02: disable CiR process B#16#03: conditionally disable CiR process
FRZ_TIME	INPUT	TIME	Upper limit of CiR synchronization time Default setting: T#1000 ms Permissible range: T#200ms to T#2500ms
RET_VAL	RETURN	INT	Error information
A_FT	OUTPUT	TIME	Currently valid upper limit of CiR synchronization time

21 Interrupt Handling

Interrupt handling is always event-driven. When such an event occurs, the operating system interrupts scanning of the main program and calls the routine allocated to this particular event. When this routine has executed, the operating system resumes scanning of the main program at the point of interruption. Such an interruption can take place after every operation (statement).

Applicable events may be interrupts and errors. The order in which virtually simultaneous interrupt events are handled is regulated by a priority scheduler. Each event has a particular servicing priority. Several interrupt events can be combined into priority classes.

Every routine associated with an interrupt event is written in an organization block in which additional blocks can be called. A higher-priority event interrupts execution of the routine in an organization block with a lower priority. You can affect the interruption of a program by high-priority events using system functions.

21.1 General Remarks

SIMATIC S7 provides the following interrupt events (interrupts):

- Time-of-day interrupt
  An interrupt generated by the operating system at a specific time of day, either once only or periodically
- Time-delay interrupt
  An interrupt generated after a specific amount of time has passed; a system function call determines the instant at which this time period begins
- Watchdog interrupt
  An interrupt generated by the operating system at periodic intervals
- Hardware interrupt
  An interrupt from a module, either via an input derived from a process signal or generated on the module itself
- DPV1 interrupt
  An interrupt from a PROFIBUS DPV1 slave
- Multiprocessor interrupt
  An interrupt generated by another CPU in a multiprocessor network
- Synchronous cycle interrupt
  An interrupt from a PROFIBUS DP master synchronous to the DP cycle

Other interrupt events are the synchronous errors which may occur in conjunction with program scanning and the asynchronous errors, such as diagnostic interrupts. The handling of these events is discussed in 23, "Error Handling".

Priorities

An event with a higher priority interrupts a program being processed with lower priority because of another event. The main program has the lowest priority (priority class 1), asynchronous errors the highest (priority class 26), apart from the start-up routine. All other events are in the intervening priority classes. In S7-300 systems, the priorities are fixed; in S7-400 systems, you can change the priorities by parameterizing the CPU accordingly.

An overview of all priority classes, together with the default organization blocks for each, is presented in Chapter 3.1.2, "Priority Classes".

Disabling interrupts

The organization blocks for event-driven program scanning can be disabled and enabled with system functions SFC 39 DIS_IRT and SFC 40 EN_IRT and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT (see Chapter 21.9, "Handling Interrupt Events").

Current signal states

In an interrupt handling routine, one of the requirements is that you work with the current signal states of the I/O modules (and not with the signal states of the inputs that were updated at the start of the main program) and write the fetched signal states direct to the I/O (not waiting until the process-image output table is updated at the end of the main program).

In the case of a few inputs and outputs for the interrupt handling routine, it is enough to access the I/O modules direct with load and transfer operations (AWL) or with the MOVE box (LAD, FBD). You are recommended here to maintain a strict separation between the main program and the interrupt handling routine with regard to the I/O signals.

If you want to process many input and output signals in the interrupt handling routine, the solution on the S7-400 CPUs is to use subprocess images. When assigning addresses, you assign each module to a subprocess image. With SFC 26 UPDAT_PI and SFC 27 UPDAT_PO, you update the subprocess images in the user program (see also Chapter 20.2.1, "Process Image Updating").

On new S7-400 CPUs, you can assign an input and an output subprocess image to each interrupt organization block (each interrupt priority class) and so cause the process images to be updated automatically when the interrupt occurs.

Start information, temporary local data

Every organization block delivers the start information in the first 20 bytes of its temporary local data. You can generate the declaration of the start information using own data yourself, or you can use the templates from the *Standard Library* under *Organization Blocks*.

In S7-300 systems, the available temporary local data have a fixed length of 256 bytes per priority class. In S7-400 systems, you can specify the length per priority class by parameterizing the CPU accordingly (parameter block "local data"), whereby the total may not exceed a CPU-specific maximum. Note that the minimum number of bytes for temporary local data for the priority class used must be 20 bytes so as to be able to accommodate the start information. Specify zero for unused priority classes.

Note that you can only directly read the start information of an organization block in the block itself, since temporary local data are involved. If you also require values from the start information in blocks which are located in lower call levels, call the system function SFC 6 RD_SINFO at the corresponding point in the program (see Chapter 20.2.5, "Start Information").

Actual interrupt information

Bytes 4 to 11 of the start information in the interrupt organization block contain the information specific to the triggered interrupt. In many cases, the components triggering the interrupt deliver additional information which you can then read in the interrupt organization block using the system function block SFB 54 RALRM (see Chapter 21.9.3, "Reading additional Interrupt Information").

21.2 Time-of-Day Interrupts

Time-of-day interrupts are used when you want to run a program at a particular time, either once only or periodically, for instance daily. In STEP 7, organization blocks OB 10 to OB 17 are provided for servicing time-of-day interrupts; which of these eight organization blocks are actually available depends on the CPU used.

You can configure the time-of-day interrupts in the hardware configuration data or control them at runtime via the program using system functions. The prerequisite for proper handling of the time-of-day interrupts is a correctly set real-time clock on the CPU.

Table 21.1 shows the start information for the time-of-day interrupts. The dummy value xx represents the number of the associated interrupt organization block 10 to 17.

21.2.1 Handling Time-of-Day Interrupts

General remarks

To start a time-of-day interrupt, you must first set the start time, then activate the interrupt. You can perform the two activities separately via the hardware configuration data or using SFCs. Note that when activated via the hardware configuration data, the time-of-day interrupt is started automatically following parameterization of the CPU.

You can start a time-of-day interrupt in two ways:

- ▷ Single-shot: the relevant OB is called once only at the specified time, or
- ▷ Periodically: depending on the parameter assignments, the relevant OB is started every minute, hourly, daily, weekly, monthly or yearly.

Following a single-shot time-of-day interrupt OB call, the time-of-day interrupt is canceled. You can also cancel a time-of-day interrupt with SFC 29 CAN_TINT.

If you want to once again use a canceled time-of-day interrupt, you must set the start time again, then reactivate the interrupt.

You can query the status of a time-of-day interrupt with SFC 31 QRY_TINT.

Performance characteristics during startup

During a cold restart or warm restart, the operating system clears all settings made with SFCs. Settings made via the hardware configuration data are retained. On a hot restart, the CPU resumes servicing of the time-of-day interrupts in the first complete scan cycle of the main program.

You can query the status of the time-of-day interrupts in the start-up OB by calling SFC 31, and subsequently cancel or re-set and reactivate the interrupts. The time-of-day interrupts are serviced only in RUN mode.

Performance characteristics on error

If a time-of-day interrupt OB is called but was not programmed, the operating system calls OB 85 (program execution error). If OB 85 was not programmed, the CPU goes to STOP.

Time-of-day interrupts that were deselected when the CPU was parameterized cannot be serviced, even when the relevant OB is available. The CPU goes to STOP.

If you activate a time-of-day interrupt on a single-shot basis, and if the start time has already passed (from the real-time clock's point of view), the operating system calls OB 80 (timing

Table 21.1 Start Information for Time-of-Day Interrupts

Byte	Variable Name	Data Type	Description	Contents
0	OBxx_EV_CLASS	BYTE	Event class	B#16#11 = UP event
1	OBxx_STRT_INF	BYTE	Start request for the interrupt OB	B#16#11 = OB 31
2	OBxx_PRIORITY	BYTE	Priority class	Default value 2 for all time-of-day interrupts
3	OBxx_OB_NUMBR	BYTE	OB number	B#16#xx
4	OBxx_RESERVED_1	BYTE	Reserved	-
5	OBxx_RESERVED_2	BYTE	Reserved	-
6..7	OBxx_PERIOD_EXE	WORD	Interval with OBs called periodically	See description of SFC 28 SET_TINT
8..9	OBxx_RESERVED_3	INT	Reserved	-
10..11	OBxx_RESERVED_4	INT	Reserved	-
12..19	OBxx_DATE_TIME	DATE_AND_TIME	Occurrence of event	Called time of OB

xx represents the OB numbers 10 to 17

error). If OB 80 is not available, the CPU goes to STOP.

If you activate a time-of-day interrupt on a periodic basis, and if the start time has already passed (from the real-time clock's point of view), the time-of-day interrupt OB is executed the next time that time period comes due.

If you set the real-time clock ahead by more than approx. 20 s, whether for the purpose of correction or synchronization, thus skipping over the start time for the time-of-day interrupt, the operating system calls OB 80 (timing error). The time-of-day interrupt OB is then executed precisely once.

If you set the real-time clock back by more than approx. 20 s, whether for the purpose of correction or synchronization, an activated time-of-day interrupt OB will no longer be executed at the instants which are already past.

If a time-of-day interrupt OB is still executing when the next (periodic) call occurs, the operating system invokes OB 80 (timing error). When OB 80 and the time-of-day interrupt OB have executed, the time-of-day interrupt OB is restarted.

Disabling, delaying and enabling

Time-of-day interrupt OB calls can be disabled and enabled with SFC 39 DIS_IRT and SFC 40 EN_IRT, and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT.

21.2.2 Configuring Time-of-Day Interrupts with STEP 7

The time-of-day interrupts are configured via the hardware configuration data. Open the selected CPU with EDIT → OBJECT PROPERTIES and choose the "Time-of-Day" tab from the dialog box.

In S7-300 controllers, the processing priority is permanently set to 2. In S7-400 controllers, you can set a priority between 2 and 24, depending on the CPU, for each possible OB; priority 0 deselects an OB. You should not assign a priority more than once, as interrupts might be lost when more than 12 interrupt events with the same priority occur simultaneously.

The "Active" option activates automatic starting of the time-of-day interrupt. The "Execution" option screens a list which allows you to choose whether you want the OB to execute on a single-shot basis or at specific intervals. The final parameter is the start time (date and time).

When it saves the hardware configuration, STEP 7 writes the compiled data to the *System Data* object in the offline user program *Blocks*. From here, you can load the parameter assignment data into the CPU while the CPU is at STOP; these data then go into force immediately.

21.2.3 System Functions for Time-of-Day Interrupts

The following system functions can be used for time-of-day interrupt control:

- ▷ SFC 28 SET_TINT
  Set time-of-day interrupt
- ▷ SFC 29 CAN_TINT
  Cancel time-of-day interrupt
- ▷ SFC 30 ACT_TINT
  Activate time-of-day interrupt
- ▷ SFC 31 QRY_TINT
  Query time-of-day interrupt

The parameters for these system functions are listed in Table 21.2.

SFC 28 SET_TINT
Set time-of-day interrupt

You determine the start time for a time-of-day interrupt by calling system function SFC 28 SET_TINT. SFC 28 sets only the start time; to start the time-of-day interrupt OB, you must activate the time-of-day interrupt with SFC 30 ACT_TINT. Specify the start time in the SDT parameter in the format DATE_AND_TIME, for instance DT#1997-06-30-08:30. The operating system ignores seconds and milliseconds and sets these values to zero. Setting the start time will overwrite the old start time value, if any. An active time-of-day interrupt is canceled, that is, it must be reactivated

Table 21.2 SFC Parameters for Time-of-Day Interrupts

SFC	Parameter	Declaration	Data Type	Contents, Description
28	OB_NR	INPUT	INT	Number of the OB to be called at the specified time on a single-shot basis or periodically
	SDT	INPUT	DT	Start date and start time in the format DATE_AND_TIME
	PERIOD	INPUT	WORD	Period on which start time is based: W#16#0000 = Single-shot W#16#0201 = Every minute W#16#0401 = Hourly W#16#1001 = Daily W#16#1201 = Weekly W#16#1401 = Monthly W#16#2001 = Last in the month W#16#1801 = Yearly
	RET_VAL	RETURN	INT	Error information
29	OB_NR	INPUT	INT	Number of the OB whose start time is to be deleted
	RET_VAL	RETURN	INT	Error information
30	OB_NR	INPUT	INT	Number of the OB to be activated
	RET_VAL	RETURN	INT	Error information
31	OB_NR	INPUT	INT	Number of the OB whose status is to be queried
	RET_VAL	RETURN	INT	Error information
	STATUS	OUTPUT	WORD	Status of the time-of-day interrupt

SFC 30 ACT_TINT
Activate time-of-day interrupt

A time-of-day interrupt is activated by calling system function SFC 30 ACT_TINT. When a TOD interrupt is activated, it is assumed that a time has been set for the interrupt. If, in the case of a single-shot interrupt, the start time is already past, SFC 30 reports an error. In the case of a periodic start, the operating system calls the relevant OB at the next applicable time. Once a single-shot time-of-day interrupt has been serviced, it is, for all practical purposes, canceled. You can re-set and reactivate it (for a different start time) if desired.

SFC 29 CAN_TINT
Cancel time-of-day interrupt

You can delete a start time, thus deactivating the time-of-day interrupt, with system function SFC 29 CAN_TINT. The respective OB is no longer called. If you want to use this same time-of-day interrupt again, you must first set the start time, then activate the interrupt.

SFC 31 QRY_TINT
Query time-of-day interrupt

You can query the status of a time-of-day interrupt by calling system function SFC 31 QRY_TINT. The required information is returned in the STATUS parameter.

When the bits have signal state "1", they have the following meanings:

0 TOD interrupt disabled by operating system
1 New TOD interrupt rejected
2 TOD interrupt not activated and not expired
3 (- Reserved -)
4 TOD interrupt OB loaded
5 No disable
6 (and following: - Reserved -)

21.3 Time-Delay Interrupts

A time-delay interrupt allows you to implement a delay timer independently of the standard timers. In STEP 7, organization blocks OB 20 to OB 23 are set aside for time-delay interrupts;

Table 21.3 Start Information for Time-Delay Interrupts

Byte	Variable Name	Data Type	Description	Contents
0	OBxx_EV_CLASS	BYTE	Events class	B#16#11 = UP event
1	OBxx_STRT_INF	BYTE	Start request for the interrupt OB	B#16#21 – OB 20
2	OBxx_PRIORITY	BYTE	Priority class	Default values 3 to 6 (OB 20 to OB 23)
3	OBxx_OB_NUMBR	BYTE	OB number	B#16#xx
4	OBxx_RESERVED_1	BYTE	Reserved	-
5	OBxx_RESERVED_2	BYTE	Reserved	-
6..7	OBxx_SIGN	WORD	Job identification	See description of SFC 32 SRT_DINT
8..11	OBxx_DTIME	TIME	Expired delay time	See description of SFC 32 SRT_DINT
12..19	OBxx_DATE_TIME	DATE_ AND_TIME	Occurrence of event	Call time of OB

xx represents the OB numbers 20 to 23

which of these four organization blocks are actually available depends on the CPU used.

The priorities for time-delay interrupt OBs are programmed in the hardware configuration data.

Table 21.3 shows the start information for the time-delay interrupts. The dummy value xx represents the number of the associated interrupt organization block 20 to 23.

21.3.1 Handling Time-Delay Interrupts

General remarks

A time-delay interrupt is started by calling SFC 32 SRT_DINT; this system function also passes the delay interval and the number of the selected organization block to the operating system. When the delay interval has expired, the OB is called.

You can cancel servicing of a time-delay interrupt, in which case the associated OB will no longer be called.

You can query the status of a time-delay interrupt with SFC 34 QRY_DINT.

Performance characteristics during startup

On a cold restart or warm restart, the operating system deletes all programmed settings for time-delay interrupts. On a hot restart, the settings are retained until processed in RUN mode, whereby the "residual cycle" is counted as part of the start-up routine.

You can start a time-delay interrupt in the start-up routine by calling SFC 32. When the delay interval has expired, the CPU must be in RUN mode in order to be able to execute the relevant organization block. If this is not the case, the CPU waits to call the organization block until the start-up routine has terminated, then calls the time-delay interrupt OB before the first network in the main program.

Performance characteristics on error

If no time-delay interrupt OB has been programmed, the operating system calls OB 85 (program execution error). If there is no OB 85 in the user program, the CPU goes to STOP.

If the delay interval has expired and the associated OB is still executing, the operating system calls OB 80 (timing error) or goes to STOP if there is no OB 80 in the user program.

Time-delay interrupts which were deselected during CPU parameterization cannot be serviced, even when the respective OB has been programmed. The CPU goes to STOP.

Disabling, delaying and enabling

The time-delay interrupt OBs can be disabled and enabled with system functions SFC 39 DIS_IRT and SFC 40 EN_IRT, and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT.

21.3.2 Configuring Time-Delay Interrupts with STEP 7

Time-delay interrupts are configured in the hardware configuration data. Simply open the selected CPU with EDIT → OBJECT PROPERTIES and choose the "Interrupts" tab from the dialog box.

In S7-300 controllers, the priority is permanently preset to 3. In S7-400 controllers, you can choose a priority between 2 and 24, depending on the CPU, for each possible OB; choose priority 0 to deselect an OB. You should not assign a priority more than once, as interrupts could be lost if more than 12 interrupt events with the same priority occur simultaneously.

When it saves the hardware configuration, STEP 7 writes the compiled data to the *System Data* object in the offline user program *Blocks*. From here, you can transfer the parameter assignment data while the CPU is at STOP; the data take effect immediately.

21.3.3 System Functions for Time-Delay Interrupts

A time-delay interrupt can be controlled with the following system functions:

- SFC 32 SRT_DINT
  Start time-delay interrupt
- SFC 33 CAN_DINT
  Cancel time-delay interrupt
- SFC 34 QRY_DINT
  Query time-delay interrupt

The parameters for these system functions are listed in Table 21.4.

SFC 32 SRT_DINT
Start time-delay interrupt

A time-delay interrupt is started by calling system function SFC 32 SRT_DINT. The SFC call is also the start time for the programmed delay interval. When the delay interval has expired, the CPU calls the programmed OB and passes the time delay value and a job identifier in the start information for this OB. The job identifier is specified in the SIGN parameter for SFC 32; you can read the same value in bytes 6 and 7 of the start information for the associated time-delay interrupt OB. The time delay is set in increments of 1 ms. The accuracy of the time delay is also 1 ms. Note that execution of the time-delay interrupt OB may itself be delayed

Table 21.4 SFC Parameters for Time-Delay Interrupts

SFC	Parameter	Declaration	Data Type	Contents, Description
32	OB_NR	INPUT	INT	Number of the OB to be called when the delay interval has expired
	DTIME	INPUT	TIME	Delay interval; permissible: T#1ms to T#1m
	SIGN	INPUT	WORD	Job identification in the respective OB's start information when the OB is called (arbitrary characters)
	RET_VAL	RETURN	INT	Error information
33	OB_NR	INPUT	INT	Number of the OB to be canceled
	RET_VAL	RETURN	INT	Error information
34	OB_NR	INPUT	INT	Number of the OB whose status is to be queried
	RET_VAL	RETURN	INT	Error information
	STATUS	OUTPUT	WORD	Status of the time-delay interrupt

Table 21.3 Start Information for Time-Delay Interrupts

Byte	Variable Name	Data Type	Description	Contents
0	OBxx_EV_CLASS	BYTE	Events class	B#16#11 = UP event
1	OBxx_STRT_INF	BYTE	Start request for the interrupt OB	B#16#21 = OB 20
2	OBxx_PRIORITY	BYTE	Priority class	Default values 3 to 6 (OB 20 to OB 23)
3	OBxx_OB_NUMBR	BYTE	OB number	B#16#xx
4	OBxx_RESERVED_1	BYTE	Reserved	-
5	OBxx_RESERVED_2	BYTE	Reserved	-
6..7	OBxx_SIGN	WORD	Job identification	See description of SFC 32 SRT_DINT
8..11	OBxx_DTIME	TIME	Expired delay time	See description of SFC 32 SRT_DINT
12..19	OBxx_DATE_TIME	DATE_ AND_TIME	Occurrence of event	Call time of OB

xx represents the OB numbers 20 to 23

which of these four organization blocks are actually available depends on the CPU used.

The priorities for time-delay interrupt OBs are programmed in the hardware configuration data.

Table 21.3 shows the start information for the time-delay interrupts. The dummy value xx represents the number of the associated interrupt organization block 20 to 23.

21.3.1 Handling Time-Delay Interrupts

General remarks

A time-delay interrupt is started by calling SFC 32 SRT_DINT; this system function also passes the delay interval and the number of the selected organization block to the operating system. When the delay interval has expired, the OB is called.

You can cancel servicing of a time-delay interrupt, in which case the associated OB will no longer be called.

You can query the status of a time-delay interrupt with SFC 34 QRY_DINT.

Performance characteristics during startup

On a cold restart or warm restart, the operating system deletes all programmed settings for time-delay interrupts. On a hot restart, the settings are retained until processed in RUN mode, whereby the "residual cycle" is counted as part of the start-up routine.

You can start a time-delay interrupt in the start-up routine by calling SFC 32. When the delay interval has expired, the CPU must be in RUN mode in order to be able to execute the relevant organization block. If this is not the case, the CPU waits to call the organization block until the start-up routine has terminated, then calls the time-delay interrupt OB before the first network in the main program.

Performance characteristics on error

If no time-delay interrupt OB has been programmed, the operating system calls OB 85 (program execution error). If there is no OB 85 in the user program, the CPU goes to STOP.

If the delay interval has expired and the associated OB is still executing, the operating system calls OB 80 (timing error) or goes to STOP if there is no OB 80 in the user program.

Time-delay interrupts which were deselected during CPU parameterization cannot be serviced, even when the respective OB has been programmed. The CPU goes to STOP.

21.4.1 Handling Watchdog Interrupts

Triggering watchdog interrupts in an S7-300

In an S7-300, there is a limited choice of watchdog interrupts with fixed priority, depending on the CPU. You can set the interval in the range from 1 millisecond to 1 minute, in 1-millisecond increments, by parameterizing the CPU accordingly (an interval of 500 µs on the CPU 319 from firmware version V2.6).

Triggering watchdog interrupts in an S7-400

You define a watchdog interrupt when you parameterize the CPU. A watchdog interrupt has three parameters: the interval, the phase offset, and the priority. You can set all three. Specifiable values for interval and phase offset are from 1 millisecond to 1 minute, in 1-millisecond increments; the priority may be set to a value between 2 and 24 or to zero, depending on the CPU (zero means the watchdog interrupt is not active).

STEP 7 provides the organization blocks listed in Table 21.6, in their maximum configurations.

Table 21.6 Defaults for Watchdog Interrupts

OB	Time Interval	Phase	Priority
30	5 s	0 ms	7
31	2 s	0 ms	8
32	1 s	0 ms	9
33	500 ms	0 ms	10
34	200 ms	0 ms	11
35	100 ms	0 ms	12
36	50 ms	0 ms	13
37	20 ms	0 ms	14
38	10 ms	0 ms	15

Phase offset

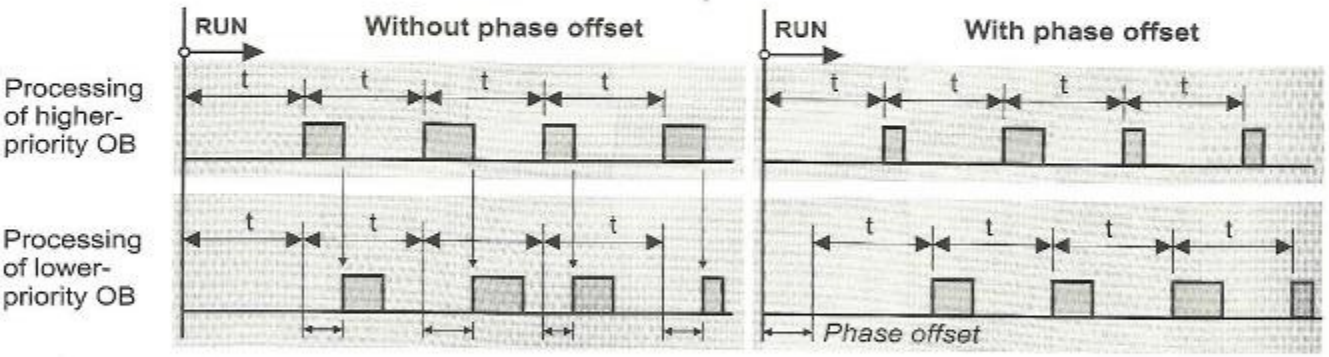
The phase offset can be used to process watchdog interrupt programs in the precise time frame even if they have the same time interval or a shared multiple. Use of the phase offset achieves a higher interval accuracy.

The start time of the time interval and the phase offset is the instant of transition from STARTUP to RUN. The call instant for a watchdog interrupt OB is thus the time interval plus the phase offset. Figure 21.1 shows an example of this. In the left section, no phase offset is set, and consequently start of processing of the lower priority organization block is delayed by the current processing time of the higher priority organization block in each case.

If, on the other hand, a phase shift is configured and it is greater than the maximum processing time of the higher-priority organization block, the lower-priority organization block is processed in the precise time frame.

Performance characteristics during startup

Watchdog interrupts cannot be serviced in the start-up OB. The time intervals do not begin until a transition is made to RUN mode.



The start of the lower-priority OB is shifted by the processing time of the higher-priority OB

With a sufficiently high phase offset, the lower-priority OB starts exactly at the time interval

Figure 21.1 Example of Phase Offset for Watchdog Interrupts

Performance characteristics on error

When the same watchdog interrupt is generated again while the associated watchdog interrupt handling OB is still executing, the operating system calls OB 80 (timing error). If OB 80 has not been programmed, the CPU goes to STOP.

The operating system saves the watchdog interrupt that was not serviced, servicing it at the next opportunity. Only one unserviced watchdog interrupt is saved per priority class, regardless of how many unserviced watchdog interrupts accumulate.

Watchdog interrupts that were deselected when the CPU was parameterized cannot be serviced, even when the corresponding OB is available. The CPU goes to STOP in this case.

Disabling, delaying and enabling

Calling of the watchdog interrupt OBs can be disabled and enabled with system functions SFC 39 DIS_IRT and SFC 40 EN_IRT and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT.

21.4.2 Configuring Watchdog Interrupts with STEP 7

Watchdog interrupts are configured via the hardware configuration data. Simply open the selected CPU with EDIT → OBJECT PROPERTIES and choose the "Cyclic Interrupt" tab from the dialog box.

In S7-300 controllers, the processing priority is permanently set to 12. In S7-400 controllers, you may set a priority between 2 and 24 for each possible OB (CPU-specific); priority 0 deselects the OB to which it is assigned. You should not assign a priority more than once, as interrupts might be lost if more than 12 interrupt events with the same priority occur simultaneously.

The interval for each OB is selected under "Execution", the delayed call instant under "Phase Offset".

When it saves the hardware configuration, STEP 7 writes the compiled data to the *System Data* object in the offline user program *Blocks*. From here, you can load the parameter assignment data into the CPU while the CPU is at STOP; the data take effect immediately.

21.5 Hardware Interrupts

Hardware interrupts are used to enable the immediate detection in the user program of events in the controlled process, making it possible to respond with an appropriate interrupt handling routine. STEP 7 provides organization blocks OB 40 to OB 47 for servicing hardware interrupts; which of these eight organization blocks are actually available, however, depends on the CPU.

Hardware interrupt handling is programmed in the hardware configuration data. With system functions SFC 55 WR_PARM, SFC 56 WR_DPARM and SFC 57 PARM_MOD, you can (re)parameterize the modules with hardware interrupt capability even in RUN mode.

Table 21.7 shows the start information for the process interrupts. The dummy value xx represents the number of the associated interrupt organization block 40 to 47.

21.5.1 Generating a Hardware Interrupt

A hardware interrupt is generated on the modules with this capability. This could, for example, be a digital input module that detects a signal from the process or a function module that generates a hardware interrupt because of an activity taking place on the module.

By default, hardware interrupts are disabled. A parameter is used to enable servicing of a hardware interrupt (static parameter), and you can specify whether the hardware interrupt should be generated for a coming event, a leaving event, or both (dynamic parameter). Dynamic parameters are parameters which you can modify at runtime using SFCs.

In an intelligent DP slave equipped for this purpose, you can initiate a process interrupt in the master CPU with SFC 7 DP_PRAL.

The hardware interrupt is acknowledged on the module when the organization block containing the service routine for that interrupt has finished executing.

Table 21.7 Start Information for Hardware Interrupts

Byte	Variable Name	Data Type	Description	Contents
0	OBxx_EV_CLASS	BYTE	Event class	B#16#11 = UP event
1	OBxx_STRT_INF	BYTE	Start request for the interrupt OB	B#16#41 = OB 41
2	OBxx_PRIORITY	BYTE	Priority class	Default values 16 to 23 (OB 40 to 47)
3	OBxx_OB_NUMBR	BYTE	OB number	B#16#xx
4	OBxx_RESERVED_1	BYTE	Reserved	-
5	OBxx_IO_FLAG	BYTE	I/O identification	B#16#54 = input module, input submodule B#16#55 = output module, output submodule
6..7	OBxx_MDL_ADDR	WORD	Module starting address of components triggering interrupt	
8..11	OBxx_POINT_ADDR	DWORD	Interrupt information	
12..19	OBxx_DATE_TIME	DATE_AND_TIME	Occurrence of event	Call time of OB

xx represents the OB numbers 40 to 47

Resolution on the S7-300

If an event occurs during execution of a hardware interrupt OB which itself would trigger generation of the same hardware interrupt, that hardware interrupt will be lost when the event that triggered it is no longer present following acknowledgment. It makes no difference whether the event comes from the module whose hardware interrupt is currently being serviced or from another module.

A diagnostic interrupt can be generated while a hardware interrupt is being serviced. If another hardware interrupt occurs on the same channel between the time the first hardware interrupt was generated and the time that interrupt was acknowledged, the loss of the latter interrupt is reported via a diagnostic interrupt to system diagnostics.

Resolution on the S7-400

If during execution of a hardware interrupt OB an event occurs on the same channel on the same module which would trigger the same hardware interrupt, that interrupt is lost. If the event occurs on another channel on the same module or on another module, the operating system restarts the OB as soon as it has finished executing.

21.5.2 Servicing Hardware Interrupts

Querying interrupt information

The start address of the module that triggered the hardware interrupt is in bytes 6 and 7 of the hardware interrupt OB's start information. If this address is an input address, byte 5 of the start information contains B#16#54; otherwise it contains B#16#55. If the module in question is a digital input module, bytes 8 to 11 contain the status of the inputs; for any other type of module, these bytes contain the interrupt status of the module.

Interrupt handling in the start-up routine

In the start-up routine, the modules do not generate hardware interrupts. Interrupt handling begins with the transition to RUN mode. Any

hardware interrupts pending at the time of the transition are lost.

Error handling

If a hardware interrupt is generated for which there is no hardware interrupt OB in the user program, the operating system calls OB 85 (program execution error). The hardware interrupt is acknowledged. If OB 85 has not been programmed, the CPU goes to STOP.

Hardware interrupts deselected when the CPU was parameterized cannot be serviced, even when the OBs for these interrupts have been programmed. The CPU goes to STOP.

Disabling, delaying and enabling

Calling of the hardware interrupt OBs can be disabled and enabled with system functions SFC 39 DIS_IRT and SFC 40 EN_IRT, and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT.

21.5.3 Configuring Hardware Interrupts with STEP 7

Hardware interrupts are programmed in the hardware configuration data. Open the selected CPU with EDIT → OBJECT PROPERTIES and choose the "Interrupts" tab in the dialog box.

In S7-300 systems, the default priority for OB 40 is fixed. In S7-400 systems, you can choose a priority between 2 and 24 for every possible OB (on a CPU-specific basis); priority 0 deselects execution of an OB. You should never assign the same priority twice because interrupts can be lost when more than 12 interrupt events with the same priority occur simultaneously.

You must also enable the triggering of hardware interrupts on the respective modules. To this purpose, these modules are parameterized much the same as the CPU.

When it saves the hardware configuration, STEP 7 writes the compiled data to the *System Data* object in offline user program *Blocks;* from here, you can load the parameterization data into the CPU while the CPU is in STOP mode. The parameterization data for the CPU go into force immediately following loading; the parameter assignment data for the modules take effect after the next start-up.

21.6 DPV1 Interrupts

PROFIBUS DPV1 slaves can trigger the following interrupts in addition to the types previously known with SIMATIC S7:

- ▷ Status interrupt, e.g. if the DPV1 slave changes its operating mode; the interrupt organization block OB 55 is called.
- ▷ Update interrupt, e.g. if the DPV1 slave has been parameterized over the PROFIBUS or directly; the interrupt organization block OB 56 is called.
- ▷ Vendor interrupt if an event envisaged by the vendor occurs in the DPV1 slave; the interrupt organization block OB 57 is called. The events which can trigger the interrupt are defined by the vendor of the DPV1 slave.

The origin of the interrupt, the interrupt specifier and the length of interrupt information additionally available are specified in the start information of the DPV1 interrupt organization blocks (Table 21.8). You can read the supplementary interrupt information using SFB 54 RALRM (see Chapter 21.9.3, "Reading additional Interrupt Information").

Performance characteristics during startup

PROFIBUS DPV1 slaves can also generate interrupts when the master CPU is at STOP. In this state, the master CPU cannot call an interrupt organization block; processing of the missed interrupts is not carried out when the CPU enters the RUN state.

However, the received interrupt events are entered into the diagnostics buffer and the module status data. You can read the module status data using the system function SFC 51 RDSYSST.

Error handling

If the corresponding DPV1 interrupt OB is missing in the user program when the DPV1 interrupt

is triggered, the operating system calls the OB 85 (program execution fault). The DPV1 interrupt is acknowledged. If the OB 85 is not present, the CPU enters the STOP state.

Table 21.8 Start Information for DPV1 Interrupts

Byte	Variable Name	Data Type	Description	Contents
0	OBxx_EV_CLASS	BYTE	Event class	B#16#11 = UP event
1	OBxx_STRT_INF	BYTE	Start request for the OB xx	B#16#xx
2	OBxx_PRIORITY	BYTE	Priority class	B#16#02 = default value
3	OBxx_OB_NUMBR	BYTE	OB number	B#16#xx
4	OBxx_RESERVED_1	BYTE	Reserved	-
5	OBxx_IO_FLAG	BYTE	I/O identification	B#16#54 = input module, input submodule B#16#55 = output module, output submodule
6..7	OBxx_MDL_ADDR	WORD	Module starting address of component triggering the interrupt	
8	OBxx_LEN	BYTE	Length of interrupt data record	
9	OBxx_TYPE	BYTE	Identification of interrupt type	B#16#00 = reserved B#16#01 = diagnostic interrupt B#16#02 = hardware interrupt B#16#03 = removal interrupt B#16#04 = insertion interrupt B#16#05 = status interrupt B#16#06 = update interrupt B#16#07..1F = reserved B#16#20..7E = vendor interrupt B#16#7F = reserved
10	OBxx_SLOT	BYTE	Slot number of component triggering the interrupt	
11	OBxx_SPEC	BYTE	Specifier	Bits 1 and 0: 0 0 reserved 0 1 UP event 1 0 DOWN event with error 1 1 DOWN event with further errors Bit 2: 0 no additional acknowledgment required 1 additional acknowledgment required Bits 3 to 7: reserved
12..19	OBxx_DATE_TIME	DATE_AND_TIME	Occurrence of event	Call time of OB

xx represents the OB numbers 55, 56 or 57

Disabling, delaying and enabling

Calling the DPV1 interrupt OBs can be disabled and enabled using SFC 39 DIS_IRT and SFC 40 EN_IRT respectively, and delayed and enabled using SFC 41 DIS_AIRT and SFC 42 EN_AIRT respectively.

Configuring DPV1 interrupts with STEP 7

The DPV1 interrupts are configured using the Hardware Configuration tool. Open the selected CPU using EDIT → OBJECT PROPERTIES, and select the "Interrupts" tab in the properties window which is then displayed.

The default priority is 2. You can set the priority between 2 and 24. Priority 0 deselects the interrupt. DPV1 interrupts which have been deselected cannot be executed, even if the corresponding OB is present. The CPU then enters the STOP status.

You must additionally parameterize interrupt triggering on the corresponding DPV1 slaves.

When saving the hardware configuration, STEP 7 writes the compiled data into the *System data* object in the offline user program *Blocks*; from here, you can download the parameterization data to the CPU when in the STOP status. The parameterization data for the CPU are immediately effective following downloading, those for the DPV1 slaves following the next startup.

21.7 Multiprocessor Interrupt

The multiprocessor interrupt allows a synchronous response to an event in all CPUs in multiprocessor mode. A multiprocessor interrupt is triggered using SFC 35 MP_ALM. Organization block OB 60, which has a fixed priority of 25, is the OB used to service a multiprocessor interrupt.

Table 21.9 shows the assignments of the start information for the multi-processor interrupt.

General remarks

An SFC 35 MP_ALM call initiates execution of the multiprocessor interrupt OB. If the CPU is in single-processor mode, OB 60 is started immediately. In multiprocessor mode, OB 60 is started simultaneously on all participating CPUs, that is to say, even the CPU in which SFC 35 was called waits before calling OB 60 until all the other CPUs have indicated that they are ready.

Table 21.9 Start Information for the Multi-processor Interrupt

Byte	Variable Name	Data Type	Description	Contents
0	OB60_EV_CLASS	BYTE	Event class	B#16#11 = UP event
1	OB60_STRT_INF	BYTE	Start request for the OB 60	B#16#61: multi-processor interrupt triggered by own CPU B#16#62: multi-processor interrupt triggered by another CPU
2	OB60_PRIORITY	BYTE	Priority class	B#16#19 = default value (25dec)
3	OB60_OB_NUMBR	BYTE	OB number	B#16#3C (60dec)
4	OB60_RESERVED_1	BYTE	Reserved	-
5	OB60_RESERVED_2	BYTE	Reserved	-
6..7	OB60_JOB	INT	Job identification	Input variable JOB of SFC 35 MP_ALM
8..9	OB60_RESERVED_3	INT	Reserved	-
10..11	OB60_RESERVED_4	INT	Reserved	-
12..19	OB60_DATE_TIME	DATE_AND_TIME	Occurrence of event	Call time of OB

Table 21.10 Parameters for SFC 35 MP_ALM

Parameter	Declaration	Data Type	Contents, Description
JOB	INPUT	BYTE	Job identification in the range B#16#00 to B#16#0F
RET_VAL	RETURN	INT	Error information

The multiprocessor interrupt is not programmed in the hardware configuration data; it is already present in every CPU with multicomputing capability. Despite this fact, however, a sufficient number of local data bytes (at least 20) must still be reserved in the CPU's "Local Data" tab under priority class 25.

Performance characteristics during startup

The multiprocessor interrupt is triggered only in RUN mode. An SFC 35 call in the start-up routine terminates after returning error 32 929 (W#16#80A1) as function value.

Performance characteristics on error

If OB 60 is still in progress when SFC 35 is recalled, the system function returns error code 32 928 (W#16#80A0) as function value. OB 60 is not started in any of the CPUs.

The unavailability of OB 60 in one of the CPUs at the time it is called or the disabling or delaying of its execution by system functions has no effect, nor does SFC 35 report an error.

Disabling, delaying and enabling

The multiprocessor OB can be disabled and enabled with system functions SFC 39 DIS_IRT and SFC 40 EN_IRT respectively, and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT respectively.

SFC 35 MP_ALM Multiprocessor interrupt

A multiprocessor interrupt is triggered with system function SFC 35 MP_ALM. Its parameters are listed in Table 21.10.

The JOB parameter allows you to forward a job identifier. The same value can be read in bytes 6 and 7 of OB 60's start information in all CPUs.

21.8 Synchronous Cycle Interrupts

Reference is made to isochrone mode if a program is executed in synchronism with the PROFIBUS DP cycle. In conjunction with constant bus cycle times, othis results in reproducible response times of equal length to the I/O. The user program executed in isochrone mode is present in the organization blocks OB 61 to OB 64. The system functions SFC 126 SYNC_PI and SFC 127 SYNC_PO are available for updating of the process image in isochrone mode.

You configure isochrone mode using the Hardware Configuration tool (see section, "Configuring constant bus cycle times and isochrone mode" in Chapter 20.4.3, "Special Functions for PROFIBUS DP"). Table 21.11 shows you the start information for the synchronous cycle interrupts. The dummy value xx represents the number of the associated interrupt organization block 61 to 64.

21.8.1 Processing the Synchronous Cycle Interrupts

The synchronous cycle interrupt is triggered by the Global Control command of the DP master.

Synchronous cycle interrupts are only processed in the operating mode RUN. An synchronous cycle interrupt in the STARTUP, STOP or HALT states is rejected. The number of OB calls which have not yet been executed is present in the start information of the isochrone mode OB when called for the first time in RUN mode.

Error handling

If an synchronous cycle interrupt arrives before the associated synchronous cycle interrupt OB has been completed, a timing error is signaled. This can occur if the user program dwells too long in an synchronous cycle interrupt OB or if

Table 21.11 Start Information for Synchronous Cycle Interrupts

Byte	Variable Name	Data Type	Description	Contents
0	OBxx_EV_CLASS	BYTE	Event class	B#16#11 = UP event
1	OBxx_STRT_INF	BYTE	Start request for the OB 60	B#16#64: OB 61, ... , B#16#67: OB 64
2	OBxx_PRIORITY	BYTE	Priority class	B#16#19 = default value (25dec)
3	OBxx_OB_NUMBR	BYTE	OB number	B#16#xx
4	OBxx_RESERVED_1	BYTE	Reserved	-
5	OBxx_RESERVED_2	BYTE	Reserved	-
6.0	OBxx_GC_VIOL	BOOL	GC violation	See text
6.1	OBxx_FIRST	BOOL	First execution following STARTUP or HALT	With "1"
7	OBxx_MISSED_EXEC	BYTE	Number of rejected OB calls	See text
8	OBxx_DP_ID	BYTE	DP master system ID of isochronous DP master system	Is configured using the Hardware Configuration tool
9	OBxx_RESERVED_3	BYTE	Reserved	-
10..11	OBxx_RESERVED_4	WORD	Reserved	-
12..19	OBxx_DATE_TIME	DT	Occurrence of event	Call time of OB

execution has been interrupted for too long as a result of program components of higher priority. The OB called by the "too early" interrupt is rejected, and the OB 80 "Timing error" is called. A reaction can then be made here to the timing error. The number of failed synchronous cycle interrupts is output in the next processed synchronous cycle interrupt OB in the start info.

In the event of an error, the DP master can omit the Global Control command (GC) or send it offset. This "GC violation" is displayed in the start information of the next synchronous cycle interrupt OB which is called correctly.

Disabling, delaying and enabling

Calling of the synchronous cycle interrupt OBs can be disabled and enabled with the system functions SFC 39 DIS_IRT and SFC 40 EN_IRT respectively, and delayed and enabled with SFC 41 DIS_AIRT and SFC 42 EN_AIRT respectively.

21.8.2 Isochrone Updating Of Process Image

The Hardware Configuration tool can be used to assign partial process images to an synchronous cycle interrupt OB. These images are not updated automatically. The system functions **SFC 126 SYNC_PI** and **SFC 127 SYNC_PO** must be used to update the inputs and outputs respectively. Updating is carried out in isochrone mode and with data consistency. The two SFCs must only be called in an synchronous cycle interrupt OB. Direct access to these partial process images should be avoided.

Table 21.12 shows the parameters of the SFC 126 SYNC_PI and SFC 127 SYNC_PO.

The partial process images are not updated if an error is detected. Exceptions:

- ▷ If an access error occurs when updating the partial process image of the inputs, the inputs of faulty modules are set to "0"; the OB 85 "Program execution error" is not called.
- ▷ A consistency warning is generated if it was not possible to transmit the complete data

Table 21.12 Parameters of the SFCs for Isochrone Updating Of Process Image

With SFC		Parameter Name	Declaration	Data Type	Contents, Description
126	127	PART	INPUT	BYTE	Number of partial process image B#16#01 to B#16#1E
126	127	RET_VAL	RETURN	INT	Error information
126	127	FLADDR	OUTPUT	WORD	In the event of an access error, the address of the first byte causing the error

consistently to the outputs. However, the data of individual slaves are consistent.

- If an access error occurs when updating the partial process image of the outputs, the data of the faulty modules are not transmitted; they remain unchanged in the partial process image. Updating of the unaffected modules is distributed between two DP cycles (consistency warning).

21.8.3 Configuration of Synchronous Cycle Interrupts with STEP 7

Configuration of synchronous cycle interrupts is carried out using the Hardware Configuration tool. Open the selected CPU with EDIT → OBJECT PROPERTIES and select the "Synchronous cycle interrupts" tab in the properties window which is then displayed.

The default priority is 25. You can set the priority between 2 and 26. Priority 0 deselects the interrupt. Synchronous cycle interrupts which have been deselected cannot be executed, even if the corresponding OB is present. In addition, you assign the isochronous DP master system and the involved partial process images to the interrupt OB.

You must also set the constant bus cycle times and isochrone mode on the corresponding DP components (see section, "Configuring constant bus cycle times and isochrone mode" in Chapter 20.4.3, "Special Functions for PROFIBUS DP").

When saving the hardware configuration, STEP 7 writes the compiled data into the object *System data* in the offline user program *Blocks*; You can download the parameterization data to the CPU from here in the STOP status. The parameterization data for the CPU are immediately effective following downloading, those for the DP components following the next startup.

21.9 Handling Interrupt Events

The system functions for disabling, delaying and enabling affect all interrupts and all asynchronous errors. System functions SFC 36 to SFC 38 are provided for handling synchronous errors.

21.9.1 Disabling and Enabling interrupts

The following system functions are available for disabling and enabling interrupts and asynchronous errors:

- SFC 39 DIS_IRT
  Disable interrupts
- SFC 40 EN_IRT
  Enable disabled interrupts

Table 21.13 lists the parameters for these system functions.

SFC 39 DIS_IRT
Disabling interrupts

System function SFC 39 DIS_IRT disables servicing of new interrupts and asynchronous errors. All new interrupts and asynchronous errors are rejected. If an interrupt or asynchronous error occurs following a Disable, the organization block is not executed; if the OB does not exist, the CPU does not go to STOP.

The Disable remains in force for all priority classes until it is revoked with SFC 40 EN_IRT. After a cold or warm restart, all interrupts and asynchronous errors are enabled.

Table 21.13 SFC Parameters for Interrupt Handling

SFC	Parameter	Declaration	Data Type	Contents, Description
39	MODE	INPUT	BYTE	Disable mode (see text)
	OB_NR	INPUT	INT	OB number (see text)
	RET_VAL	RETURN	INT	Error information
40	MODE	INPUT	BYTE	Enable mode (see text)
	OB_NR	INPUT	INT	OB number (see text)
	RET_VAL	RETURN	INT	Error information
41	RET_VAL	RETURN	INT	(New) number of delays
42	RET_VAL	RETURN	INT	Number of delays remaining

The MODE and OB_NR parameters are used to specify which interrupts and asynchronous errors are to be disabled. MODE = B#16#00 disables all interrupts and asynchronous errors. MODE = B#16#01 disables an interrupt class whose first OB number is specified in the OB_NR parameter.

For example, MODE = B#16#01 and OB_NR = 40 disables all hardware interrupts; OB = 80 would disable all asynchronous errors. MODE = B#16#02 disables the interrupt or asynchronous error whose OB number you entered in the OB_NR parameter.

Regardless of a Disable, the operating system enters each new interrupt or asynchronous error in the diagnostic buffer.

SFC 40 EN_IRT
Enabling disabled interrupts

System function SFC 40 EN_IRT enables the interrupts and asynchronous errors disabled with SFC 39 DIS_IRT. An interrupt or asynchronous error occurring after the Enable will be serviced by the associated organization block; if that organization block is not in the user program, the CPU goes to STOP (except in the case of OB 81 "Power supply errors").

The MODE and OB_NR parameters specify which interrupts and asynchronous errors are to be enabled. MODE = B#16#00 enables all interrupts and asynchronous errors. MODE = B#16#01 enables an interrupt class whose first OB number is specified in the OB_NR parameter. MODE = B#16#02 enables the interrupt or asynchronous error whose OB number you entered in the OB_NR parameter.

21.9.2 Delaying and Enabling Interrupts

The following system functions are available for delaying and enabling interrupts and asynchronous errors:

- SFC 41 DIS_AIRT
  Delay interrupts
- SFC 42 EN_AIRT
  Enable delayed interrupts

Table 21.13 lists the parameters for these system functions.

SFC 41 DIS_AIRT
Delaying Interrupts

System function SFC 41 DIS_AIRT delays the servicing of higher-priority new interrupts and asynchronous errors. Delay means that the operating system saves the interrupts and asynchronous errors which occurred during the delay and services them when the delay interval has expired. Once SFC 41 has been called, the program in the current organization block (in the current priority class) will not be interrupted by a higher-priority interrupt; no interrupts or asynchronous errors are lost.

A delay remains in force until the current OB has terminated its execution or until SFC 42 EN_AIRT is called.

You can call SFC 41 several times in succession. The RET_VAL parameter shows the number of calls. You must call SFC 42 precisely the same number of times as SFC 41 in order to reenable the interrupts and asynchronous errors.

SFC 42 EN_AIRT
Enabling delayed interrupts

System function SFC 42 EN_AIRT reenables the interrupts and asynchronous errors delayed with SFC 41. You must call SFC 42 precisely the same number of times as you called SFC 41 (in the current OB). The RET_VAL parameter shows the number of delays still in force; if RET_VAL is = 0, the interrupts and asynchronous errors have been reenabled.

If you call SFC 42 without having first called SFC 41, RET_VAL contains the value 32896 (W#16#8080).

21.9.3 Reading additional Interrupt Information

The system function block **SFB 54 RALRM** reads additional interrupt information – if present – from the components (modules or submodules) triggering the interrupt. The SFB is called in an interrupt organization block or in a block called within this. Processing of the SFB 54 RALRM is carried out synchronously, i.e. the requested data are present in the output parameters immediately following the call. Table 21.14 lists the block parameters of the SFB 54 RALRM.

The SFB 54 RALRM can basically be called in all organization blocks or execution levels for all events. If you call it in an organization block whose start event is not an interrupt from the I/O, correspondingly less information is available. Depending on the respective organization block and the components triggering the interrupt, different information is entered into the target areas specified by the TINFO and AINFO parameters (Table 21.15).

The target area TINFO (task information) contains the complete start information in bytes 0 to 19 of the organization block in which the SFB 54 RALRM was called, independent of the nesting depth in which it was called. Therefore the SFB 54 RALRM partially replaces the system function SFC 6 RD_SINFO. Administration information (e.g. which component triggered the interrupt) is present in bytes 20 to 27.

The target area AINFO (interrupt information) contains the header information (e.g. the number of received bytes of supplementary interrupt information or the type of interrupt) in bytes 0 to 3 and the component-specific additional interrupt information itself in bytes 4 to 223.

The assignment of the MODE parameter defines the operating mode of SFB 54 RALRM. With MODE = 0, the SFB shows

Table 21.14 Parameters of System Function Block SFB 54 RALRM

Parameter	Declaration	Data Type	Contents, Description
MODE	INPUT	INT	Operating mode: 0 = shows the component triggering the interrupt 1 = writes all output parameters 2 = checks whether the selected component has triggered the interrupt
F_ID	INPUT	DWORD	Module starting address of component to be scanned
MLEN	INPUT	INT	Maximum number of bytes of supplementary interrupt information to be scanned
NEW	INPUT	BOOL	TRUE = a new interrupt has been received
STATUS	OUTPUT	DWORD	Error identification
ID	OUTPUT	DWORD	Module starting address of component triggering the interrupt
LEN	OUTPUT	INT	Number of bytes of received supplementary interrupt information
TINFO	OUTPUT	ANY	Target area for OB start information and administration information
AINFO	OUTPUT	ANY	Target area for header information and supplementary interrupt information

Table 21.15 Assignment of TINFO and AINFO Parameters

Interrupt Type	OB No.	TINFO		AINFO	
		OB start information Bytes 0 to 19	Administration information Bytes 20 to 27	Header information Bytes 0 to 3 [1]	Supplementary interrupt information Bytes 4 to 223 [1]
Central hardware interrupt	40 to 47	yes	yes	yes	no
Distributed hardware interrupt	40 to 47	yes	yes	yes	As delivered by the station
Status interrupt	55	yes	yes	yes	yes
Update interrupt	56	yes	yes	yes	yes
Vendor interrupt	57	yes	yes	yes	yes
I/O redundancy error	70	yes	yes	no	no
Central diagnostic interrupt	82	yes	yes	yes	Diagnostics data record 1
Distributed diagnostic interrupt	82	yes	yes	yes	As delivered by the station
Central hot swapping interrupt	83	yes	yes	yes	no
Distributed hot swapping interrupt	83	yes	yes	yes	As delivered by the station
Rack/station failure	86	yes	yes	no	no
All other events		yes	no	no	no

With PROFINET IO:
[1] 0 to 25
[2] 26 to 1431

you the component triggering the interrupt in the ID parameter; NEW is assigned TRUE. With MODE = 1, all output parameters are written. With MODE = 2, you check whether the component specified by the F_ID parameter was the one triggering the interrupt. If this is the case, the NEW parameter has the value TRUE, and all other output parameters are written.

In order to work correctly, the SFB 54 RALRM requires its own instance data for each call in the various organization blocks, e.g. an own instance data block in each case.

22 Start-up Characteristics

22.1 General Remarks

22.1.1 Operating Modes

Before the CPU begins processing the main program following power-up, it executes a start-up routine. START-UP is one of the CPU's operating modes, as is STOP or RUN. This chapter describes the CPU's activities on a transition from and to START-UP and in the restart routine itself.

Following power-up ①, the CPU is in the STOP mode (Figure 22.1). If the mode selector on the CPU's front panel is at RUN or RUN-P, the CPU switches to START-UP mode ②, then to RUN mode ③. If an "unrecoverable" error occurs while the CPU is in START-UP or RUN mode or if you position the mode selector to STOP, the CPU returns to the STOP mode ④ ⑤.

The user program is tested with breakpoints in single-step operation in the HOLD mode. You can switch to this mode from both RUN and START-UP, and return to the original mode when you abort the test ⑥ ⑦. You can also set the CPU to the STOP mode from the HOLD mode ⑧.

When you parameterize the CPU, you can define restart characteristics with the "Restart" tab such as the maximum permissible amount of time for the Ready signals from the modules following power-up or whether the CPU is to start up when the configuration data do not coincide with the actual configuration or in what mode the CPU restart is to be in.

SIMATIC S7 has three restart modes, namely *cold restart*, *warm restart* and *hot restart*. On a cold restart or warm restart, the main program is always processed from the beginning. A hot restart resumes the main program at the point of interruption, and "finishes" the cycle.

S7 CPUs supplied before 10/98 have warm restart and hot restart.

You can scan a program on a single-shot basis in START-UP mode. STEP 7 provides organization blocks OB 102 (cold restart), OB 100 (warm restart) and OB 101 (hot restart) expressly for this purpose. Sample applications are the parameterization of modules unless this was already taken care of by the CPU, and the programming of defaults for your main program.

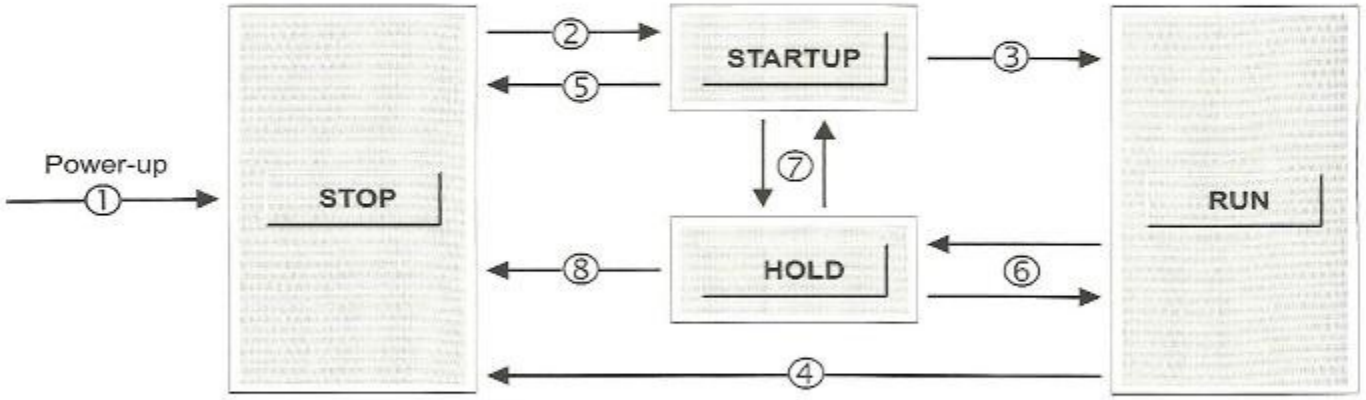


Figure 22.1 CPU Operating Modes

22.1.2 HOLD Mode

The CPU changes to the HOLD mode when you test the program with breakpoints (in "single-step mode"). The STOP LED then lights up and the RUN LED blinks.

In HOLD mode, the output modules are disabled. Writing to the modules affects the module memory, but does not switch the signal states "out" to the module outputs. The modules are not reenabled until you exit the HOLD mode.

In HOLD mode, everything having to do with timing is discontinued. This includes, for example, the processing of timers, clock memory and run-time meters, cycle time monitoring and minimum scan cycle time, and the servicing of time-of-day and time-delay interrupts. Exception: the real-time clock continues to function normally.

Every time the progression is made to the next statement in test mode, the timers for the duration of the single step run a little further, thus simulating a dynamic behavior similar to "normal" program scanning.

In HOLD mode, the CPU is capable of passive communication, that is, it can receive global data or take part in the unilateral exchange of data.

If the power fails while the CPU is in HOLD mode, battery-backed CPUs go to STOP on power recovery. CPUs without backup batteries execute an automatic warm restart.

22.1.3 Disabling the Output Modules

In the STOP and HOLD modes, modules are disabled (OD (output disable) signal). Disabled output modules output a zero signal or, if they have the capability, the replacement value. Via a variable table, you can control outputs on the modules with the "Isolate PQ" function, even in STOP mode.

During restart, the output modules remain disabled. Only when the cyclic scan begins are the output modules enabled.

On a cold restart (OB 102) and warm restart (OB 100), the process images and the module memory are cleared. If you want to scan inputs in OB 102 or in OB 100, you must load the signal states from the module using direct access. You can then set the inputs (transfer them, for instance, with load statements or with the MOVE box from address area PI to address area I), then work with the inputs.

On a warm restart, the "old" process-image input and process-image output tables, which were valid prior to power-down or STOP, are used in OB 101 and in the remainder of the cycle. At the end of that cycle, the process-image output table is transferred to module memory (but not yet switched through to the external outputs, since the output modules are still disabled).

You now have the option of parameterizing the CPU to clear the process-image output table and the module memory at the end of the warm restart. Before switching to OB 1, the CPU revokes the Disable signal so that the signal states in the module memory are applied to the external outputs.

22.1.4 Restart Organization Blocks

On a cold restart, the CPU calls organization block OB 102; on a warm restart, it calls organization block OB 100. In the absence of OB 100 or OB 102, the CPU begins cyclic program execution immediately.

On a warm restart, the CPU calls organization block OB 101 on a single-shot basis before processing the main program. If there is no OB 101, the CPU begins scanning at the point of interruption.

The start information in the temporary local data has the same format for the restart organization blocks; Table 22.1 shows the start information for OB 100. The reason for the restart is shown in the restart request (Byte 1):

B#16#81 Manual warm restart (OB 100)

B#16#82 Automatic warm restart (OB 100)

B#16#83 Manual hot restart (OB 101)

B#16#84 Automatic hot restart (OB 101)

B#16#85 Manual cold restart (OB 102)

B#16#86 Automatic cold restart (OB 102)

The number of the stop event and the additional information define the restart more precisely (tells you, for example, whether a manual warm restart was initiated via the mode selector). With this information, you can develop an appropriate event-related restart routine.

Note that no asynchronous system blocks can be processed in the startup program of an S7-300 CPU. You can set or reset outputs in the process image in the startup program, but the transmission to the output modules only takes place when transferring to RUN mode.

22.2 Power-Up

22.2.1 STOP Mode

The CPU goes to STOP in the following instances:

- ▷ When the CPU is switched on
- ▷ When the mode selector is set from RUN to STOP
- ▷ When an "unrecoverable" error occurs during program scanning
- ▷ When system function SFC 46 STP is executed
- ▷ When requested by a communication function (stop request from the programming device or via communication function blocks from another CPU)

The CPU enters the reason for the STOP in the diagnostic buffer. In this mode, you can also read the CPU information with a programming device in order to localize the problem.

In STOP mode, the user program is not scanned. The CPU retrieves the settings – either the values which you entered in the hardware configuration data when you parameterized the CPU or the defaults – and sets the modules to the specified initial state.

In STOP mode, the CPU can receive global data via GD communication and carry out passive unilateral communication functions. The real-time clock keeps running.

You can parameterize the CPU in STOP mode, for instance you can also set the MPI address, transfer or modify the user program, and execute a CPU memory reset.

22.2.2 Memory Reset

A memory reset sets the CPU to the "initial state". You can initiate a memory reset with a programming device only in STOP mode or with the mode selector: hold the switch in the MRES position for at least 3 seconds then release, and after a maximum of 3 seconds hold it in the MRES position again for at least 3 seconds.

The CPU erases the entire user program both in work memory and in RAM load memory. System memory (for instance bit memory, timers and counters) is also erased, regardless of retentivity settings. With a micro memory card, the contents of the load memory are retained during a memory reset.

Table 22.1 Start Information for the Restart OBs

Byte	Name	Data Type	Description	Contents
0	OB100_EV_CLASS	BYTE	Event class	B#16#13
1	OB100_STRTUP	BYTE	Restart request (see text)	B#16#8x (see text)
2	OB100_PRIORITY	BYTE	Priority class	Default value 27
3	OB100_OB_NUMBR	BYTE	OB number	100, 101 or 102
4	OB100_RESERVED_1	BYTE	Reserved	-
5	OB100_RESERVED_2	BYTE	Reserved	-
6..7	OB100_STOP	WORD	Number of the stop event	(see Instruction Manual)
8..11	OB100_STRT_INFO	DWORD	Additional information on the current restart	(see Instruction Manual)
12..19	OB100_DATE_TIME	DT	Date and time event occurred	Call time of the OB

The CPU sets the parameters for all modules, including its own, to their default values. The MPI parameters of the first interface are an exception. They are not changed so that a CPU whose memory has been reset can still be addressed on the MPI bus. A memory reset also does not affect the diagnostic buffer, the real-time clock, or the run-time meters.

If a micro memory card or a memory card with Flash EPROM is inserted, the CPU copies the user program from the memory card to work memory. The CPU also copies any configuration data it finds on the memory card.

22.2.3 Restoring the factory settings

In the case of newer CPUs, you can restore the factory settings with "Reset to factory settings". Proceed as follows:

- ▷ Switch off the supply voltage and remove the micro memory card or the memory card.
- ▷ Hold the mode selector switch in the MRES position and switch the supply voltage on again.
- ▷ If the LEDs SF (S7-300) or INTF (S7-400), FRCE, RUN and STOP flash slowly, release the mode selector switch again, set it to MRES again within 3 s, and hold it in this position.
- ▷ Wait until only the SF or INTF LED flashes. During this time period (approx. 5 s), you can abort the reset procedure by releasing the mode selector.
- ▷ If the SF or INTF LED shows a continuous light, release the mode selector switch.

The CPU starts up without battery backing and all LEDs light up. It executes a memory reset, then sets the MPI address to 2 and the MPI data rate to 187.5 kbit/s. As well as the memory reset, the real-time clock is set to the start date and the operating hours counter and diagnostics buffer are deleted. Following this, the CPU enters the event "Reset to factory settings" in the diagnostics buffer, and goes to STOP mode.

22.2.4 Retentivity

A memory area is retentive when its contents are retained even when the mains power is switched off as well as on a transition from STOP to RUN following power-up. In the case of the current S7-300 CPUs, retentivity is implemented with a micro memory card. In the case of the S7-400 CPUs, battery backup is required for retentivity.

Retentive memory areas may be those for bit memory, timers, counters and data blocks. The length of the retentive areas depends on the CPU. You can specify the number of retentive memory bytes, timers and counters via the "Retentivity" tab when you parameterize the CPU.

The contents of data blocks in the RAM can also be retentive. The available retentive area is CPU-specific. You can define the retentivity of a data block with the block property *Non-Retain* (see Chapter 3.2.3, "Block Properties").

With the S7-300 with micro memory card, the bit memories, timers and counters set as retentive as well as the user program and user data are saved on the micro memory card, where they are retentive even without a battery backup. With a warm restart, the non-retentive bit memories, timers and counters are deleted. The contents of data blocks declared as "non-retentive" are initialized during a warm restart (loaded with the initial values from load memory) or set to zero if a load memory object is not present.

With the S7-400, a battery backup is required for retentivity. A cold restart deletes all address areas, and loads the user program and the (configured) user data from load memory into work memory. With a warm restart, the values of the bit memories, timers and counters set as retentive are retained; the user program and the user data remain unchanged.

22.2.5 Restart Parameterization

On the "Restart" tab of the CPUs, you can affect a restart with the following settings:

- ▷ Restart when the set configuration is not the same as the actual configuration

A restart is executed even if the parameterized hardware configuration does not agree with the actual configuration. Exception: the configured PROFIBUS DP interfaces must always be present and ready for operation.

- Reset outputs on hot restart
  The S7-400 CPUs delete all process-image output tables and all peripheral outputs during a hot restart.
- Disable hot restart at manual restart
  A hot restart based on manual input or communications job is not permissible.
- Restart following POWER UP
  Definition of the type of restart following power up
- Monitoring time for ready signal of the modules
  If the monitoring time for a module times out, it is considered to be non-existent. The response of the CPU then depends on the setting "Startup when set configuration not same as actual configuration". The result is entered in the diagnostics buffer. This monitoring time is important for switching on the power on expansion racks or distributed I/O.
- Monitoring time for transferring the parameters to the modules
  If the monitoring time for a module times out, it is considered to be non-existent. The response of the CPU then depends on the setting "Startup when set configuration not same as actual configuration". The event is entered in the diagnostics buffer. (In the event of this error, you can only parameterize the CPU with a higher monitoring time – without memory reset – if you transfer the system data of an "empty" project in which the new value of the monitoring time is entered, so that the module parameterization is completed within the "old" monitoring time.)
- Monitoring time for hot restart
  If the time between power off and power on or the time between STOP and RUN is greater than the monitoring time, a hot restart is not carried out. The specification 0 ms switches the monitor off.

22.3 Types of Restart

22.3.1 START-UP Mode

The CPU executes a restart in the following cases:

- When the mains power is switched on
- When the mode selector is set (key lock switch: turn from STOP to RUN or RUN-P), or the toggle switch changed (switch from STOP to RUN)
- On a request from a communication function (initiated from a programming device or via communication function blocks from another CPU)

A *manual* restart is initiated via the mode selector or a communication function, an *automatic* restart by switching on the mains power.

The restart routine may be as long as required, and there is no time limit on its execution; the scan cycle monitor is not active.

During the execution of the restart routine, no interrupts will be serviced. Exceptions are errors that are handled as in RUN (call of the relevant error organization blocks).

In the restart routine, the CPU updates the timers, the run-time meters and the real-time clock.

During restart, the output modules are disabled, i.e., output signals cannot be transmitted. The output disable is only revoked at the end of the restart and prior to starting the cyclic program.

A restart routine can be aborted, for instance when the mode selector is actuated or when there is a power failure. The aborted restart routine is then executed from the beginning when the power is switched on. If a cold restart or warm start is aborted, it must be executed again. If a hot restart is aborted, all restart types are possible.

Figure 22.2 shows the activities carried out by an S7-400 CPU during a restart.

22.3.2 Cold Restart

On a cold restart, the CPU sets both itself and the modules to the programmed initial state,

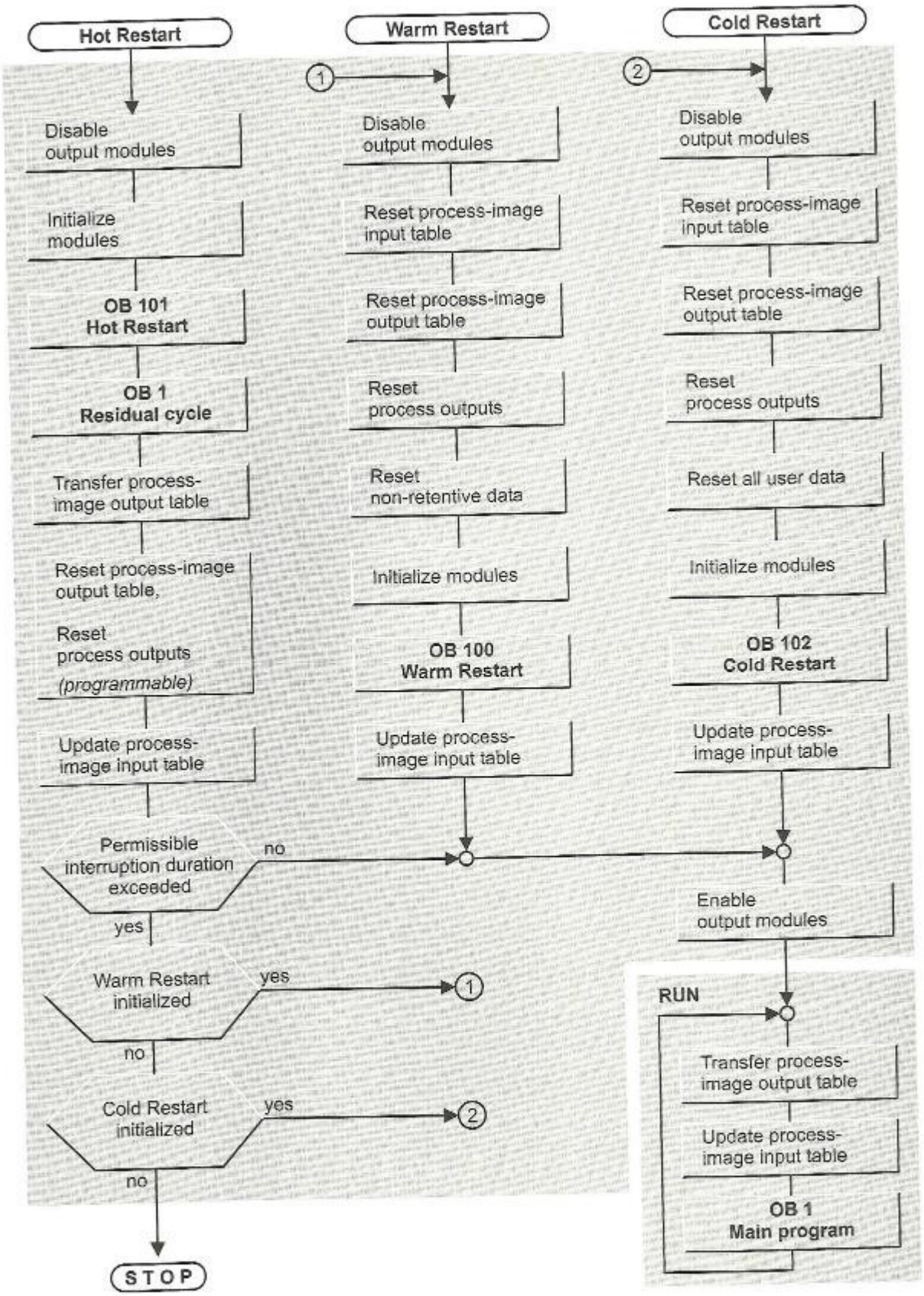


Figure 22.2 CPU Activities During Restart (S7-400)

deletes all data in the system memory (including the retentive data), calls OB 102, and then executes the main program in OB 1 from the beginning.

The current program and the current data in work memory are deleted and with them also the data blocks generated by a system function; the program from load memory is reloaded. (In contrast to memory reset, a RAM load memory is not deleted.)

Manual cold restart

With newer CPUs, a cold restart can no longer be triggered manually using the mode selector. With older CPUs, a manual cold restart is triggered using the mode selector if the switch is held in the MRES position for at least 3 s before transferring from STOP to RUN or RUN-P.

A manual cold restart can also be triggered by a communications function from a programming device or by a system block from another CPU. In this case, the mode switch must be at RUN or RUN-P.

A manual cold restart can always be initiated unless the CPU requests a memory reset.

Automatic cold restart

An automatic cold restart is initiated by switching on the mains power. The cold restart is executed if

- ▷ the CPU was not at STOP when the power was switched off
- ▷ the mode selector is at RUN or RUN-P
- ▷ the CPU was interrupted by a power outage while executing a cold restart
- ▷ "Cold restart" on "start following POWER UP" is parameterized

When operated without a backup battery, the CPU executes an automatic non-retentive warm restart. The CPU starts the memory reset automatically, then copies the user program from the memory card to work memory. The memory card must be a Flash EPROM.

22.3.3 Warm Restart

On a warm restart, the CPU sets both itself and the modules to the programmed initial state, erases the non-retentive data in the system memory, calls OB 100, and then executes the main program in OB 1 from the beginning.

The current program and the current data in work memory are retained, and also the data blocks generated per SFC.

Manual warm restart

A manual warm restart is initiated in the following instances:

- ▷ Via the mode selector on the CPU on a transition from STOP to RUN or RUN-P (on S7-400 CPUs with restart type switch, this is in the CRST position)
- ▷ Via a communication function from a PG or with an SFB from another CPU; the mode selector must be in the RUN or RUN-P position.

A manual warm restart can always be initiated unless the CPU requests a memory reset.

Automatic warm restart

An automatic warm restart is initiated by switching on the mains power. The warm restart is executed if

- ▷ the CPU was not at STOP when the power was switched off
- ▷ the mode selector is at RUN or RUN-P
- ▷ the CPU was interrupted by a power outage while executing a warm restart
- ▷ "Complete restart (warm restart)" on "start following POWER UP" is parameterized

If there is a restart type switch, it remains without effect in the case of automatic warm restart.

If the CPU contains a micro memory card, it reacts exactly like a CPU with backup battery. When operated without a micro memory card and without a backup battery, the CPU executes an automatic non-retentive warm restart. The CPU starts the memory reset automatically, then copies the user program from the memory

card to work memory. The memory card must be a Flash EPROM.

22.3.4 Hot Restart

A hot restart is possible only on an S7-400.

On a STOP or power outage, the CPU saves all interrupts as well as the internal CPU registers that are important to the processing of the user program. On a hot restart, it can therefore resume at the location in the program at which the interruption occurred. This may be the main program, or it may be an interrupt or error handling routine. All ("old") interrupts are saved and will be serviced.

The so-called "residual cycle", which extends from the point at which the CPU resumes the program following a hot restart to the end of the main program, counts as part of the restart. No (new) interrupts are serviced. The output modules are disabled, and are in their initial state.

A hot restart is permitted only when there have been no changes in the user program while the CPU was at STOP, such as modification of a block.

By parameterizing the CPU accordingly, you can specify how long the interruption may be for the CPU to still be able to execute a warm restart (from 100 milliseconds to 1 hour). If the interruption is longer, only a cold or warm restart is allowed. The length of the interruption is the amount of time between exiting of the RUN mode (STOP or power-down) and reentry into the RUN mode (following execution of OB 101 and the residual cycle).

Manual hot restart

A manual hot restart is initiated

- ▷ If the mode selector was at RUN or RUN-P when the CPU was switched on by moving the mode selector from STOP to RUN or RUN-P when the restart switch is at WRST (only possible on CPUs with restart type switch)
- ▷ Via a communications function from a programming device or with a system block (SFB) from another CPU; the mode selector must be at RUN or RUN-P.

A manual hot restart is possible only when the hot restart disable was revoked in the "Restart" tab when the CPU was parameterized. The cause of the STOP must have been a manual activity, either via the mode selector or through a communication function; only then can a manual warm restart be executed while the CPU is at STOP.

Automatic hot restart

An automatic hot restart is initiated by switching on the mains power. The CPU executes an automatic hot restart only in the following instances:

- ▷ If it was not at STOP when switched off
- ▷ If the mode selector was at RUN or RUN-P when the CPU was switched on
- ▷ "Hot restart" on "start following POWER UP" is parameterized
- ▷ If the backup battery is inserted and in working order

The position of the restart switch is irrelevant to an automatic hot restart.

22.4 Ascertaining a Module Address

Signal modules, or to be more exact the user data on input/output modules, are addressed in two manners: you use the *logical address* in the user program to address the inputs and outputs. This corresponds to the absolute address, and can be made easier to read by using symbols. The smallest logical addresses is the base address or module starting address. The CPU uses the *geographical address* to address the modules. You require the geographical address if you wish to find out the slot number of the module. This applies similarly to the user data on stations of the distributed I/O.

You can use the following system blocks to ascertain the geographical address from the logical address and vice versa:

- SFC 70 GEO_LOG
  Ascertain logical base address
- SFC 5 GADR_LGC
  Ascertain logical address of a module channel
- SFC 50 RD_LGADR
  Ascertain all logical addresses of a module
- SFC 71 LOG_GEO
  Ascertain geographical address
- SFC 49 LGC_GADR
  Ascertain slot address of a module

Table 22.2 shows the parameters for these system blocks.

The SFC 5 GADR_LGC, 49 LGC_GADR and 50 RD_LGADR have IOID and LADDR as common parameters for the logical address (= address in the I/O area). IOID is either B#16#54, which stands for the peripheral inputs (PIs) or B#16#55, which stands for the peripheral outputs (PQs). LADDR contains an I/O address in the PI or PQ area which corresponds to the specified channel. If the channel is 0, it is the module start address.

With the SFCs 70 GEO_LOG and 71 LOG_GEO, the logical address is on its own in the LADDR parameter. A differentiation is made in bit 15 as to whether the address is assigned to an input (= 0) or an output (= 1).

For the addresses determined using these system blocks, an assignment must have been made using the Hardware Configuration tool between logical address (module starting address) and slot address (location of module in a rack or in a station of the distributed I/O).

SFC 70 LOG_GEO
Ascertain the logical base address

System function SFC 70 LOG_GEO returns the logical address of a module or station. The assignment of the MASTER parameter indicates whether the station or module is inserted in a rack (central design) or whether the station is operated in a PROFIBUS or PROFINET system. Use the SLOT parameter to specify the slot number in the rack or station, and the SUBSLOT parameter for the number of the submodule. The LADDR parameter then returns the base address of the submodule. With SUBSLOT = 0, the diagnostics address of the module or station is returned.

SFC 70 GEO_LOG replaces the SFC 5 GADR_LGC, and can be used in conjunction with PROFINET IO.

SFC 5 GADR_LGC
Ascertain the logical address of a module channel

System function SFC 5 GADR_LGC returns the logical address of a channel when you specify the slot address ("geographical" address). Enter the number of the DP master system ID in the SUBNETID parameter if the module belongs to the distributed I/O or B#16#00 if the module is plugged into the central rack or an expansion rack. The RACK parameter specifies the number of the rack or, in the case of distributed I/O, the number of the station. If the module has no submodule slot, enter B#16#00 in the SUBSLOT parameter. SUBADDR contains the address offset in the module's user data (W#16#0000, for example, stands for the module start address).

SFC 71 GEO_ LOG
Ascertain the geographical address

System function SFC 71 GEO_LOG returns the geographical address of a module or station if you define the logical base address. The value in the AREA parameter defines the system in which the module is used (Table 22.3).

SFC 71 GEO_LOG replaces the SFC 49 LGC_GADR and can also be used in association with PROFINET IO.

SFC 49 LGC_GADR
Ascertain the slot address of a module

SFC 49 LGC_GADR returns the slot address of a module when you specify an arbitrary logical module address. Subtracting the address offset (parameter SUBADDR) from the specified user data address gives you the module starting address. The value in the AREA parameter specifies the system in which the module is operated (Table 22.3).

Table 22.2 Parameters for the System Blocks Used to Ascertain the Module Address

SFC	Parameter	Declaration	Data Type	Contents, Description
5	SUBNETID	INPUT	BYTE	Area identifier
	RACK	INPUT	WORD	Number of the rack
	SLOT	INPUT	WORD	Number of the slot
	SUBSLOT	INPUT	BYTE	Number of the submodule
	SUBADDR	INPUT	WORD	Offset in the module's user data address area
	RET_VAL	RETURN	INT	Error information
	IOID	OUTPUT	BYTE	Area identifier
	LADDR	OUTPUT	WORD	Logical address of the channel
50	IOID	INPUT	BYTE	Area identifier
	LADDR	INPUT	WORD	A logical module address
	RET_VAL	RETURN	INT	Error information
	PEADDR	OUTPUT	ANY	WORD field for the PI addresses
	PECOUNT	OUTPUT	INT	Number of PI addresses returned
	PAADDR	OUTPUT	ANY	WORD field for the PQ addresses
	PACOUNT	OUTPUT	INT	Number of PQ addresses returned
49	IOID	INPUT	BYTE	Area identifier
	LADDR	INPUT	WORD	A logical module address
	RET_VAL	RETURN	INT	Error information
	AREA	OUTPUT	BYTE	Area identifier
	RACK	OUTPUT	WORD	Number of the rack
	SLOT	OUTPUT	WORD	Number of the slot
	SUBADDR	OUTPUT	WORD	Offset in the module's user data address area
70	MASTER	INPUT	INT	Master system ID 0 = central I/O 1 to 31 = PROFIBUS DP 100 to 115 = PROFINET IO
	STATION	INPUT	INT	Station number; with central I/O: rack number
	SLOT	INPUT	INT	Number of the slot
	SUBSLOT	INPUT	INT	Number of the submodule
	RET_VAL	RETURN	INT	Error information
	LADDR	OUTPUT	WORD	Base address of station or module
71	LADDR	INPUT	WORD	Base address of station or module
	RET_VAL	RETURN	INT	Error information
	AREA	OUTPUT	INT	Area identifier (see Table 22.3)
	MASTER	OUTPUT	INT	Master system ID 0 = central I/O 1 to 31 = PROFIBUS DP 100 to 115 = PROFINET IO
	STATION	OUTPUT	INT	Station number; with central I/O: rack number
	SLOT	OUTPUT	INT	Number of the slot
	SUBSLOT	OUTPUT	INT	Number of the submodule
	OFFSET	OUTPUT	INT	Offset in address space of module/submodule

Table 22.3 Meaning of Output Parameters of SFC 49 LGC_GADR and SFC 71 LOG_GEO

AREA	System	Meaning of output parameters of SFC 49 LGC_GADR	Meaning of output parameters of SFC 71 LOG_GEO
0	S7-400	RACK = rack number SLOT = slot number SUBADDR = difference from base address	MASTER = 0 STATION = rack number SLOT = slot number SUBSLOT = 0 OFFSET = difference from base address
1	S7-300		
2	Distributed I/O	RACK Low byte = station number High byte = DP master system ID SLOT = slot number SUBADDR = difference from base address	With PROFIBUS DP: MASTER = DP master system ID STATION = station number SLOT = slot number SUBSLOT = 0 OFFSET = difference from base address With PROFINET IO: MASTER = PROFINET IO system ID STATION = station number SLOT = slot number SUBSLOT = subslot number OFFSET = difference from base address
3	S5 P area	RACK = rack number SLOT = slot number of adapter casing SUBADDR = address in S5 area	MASTER = 0 STATION = rack number SLOT = slot number of adapter casing SUBSLOT = 0 OFFSET = address in S5 area
4	S5 Q area		
5	S5 IM3 area		
6	S5 IM4 area		

SFC 50 RD_LGADR
Ascertain all logical addresses for a module

SFC 50 RD_LGADR returns all logical addresses for a module when you specify an arbitrary address from the user data area.

Use the PEADDR and PAADDR parameters to define an area of WORD components (a word-based ANY pointer, for example P#DBzD-BXy.x WORD nnn).

SFC 50 then shows the number of entries returned in these areas in the RECOUNT and PACOUNT parameters.

22.5 Parameterizing Modules

22.5.1 General remarks on parameterizing modules

Most S7 modules can be parameterized, that is to say, values may be set on the module which deviate from the default. To specify parameters, open the module in the hardware configuration and fill in the tabs in the dialog box. When you transfer the *System Data* object in the *Blocks* container to the PLC, you are also transferring the module parameters.

The CPU transfers the module parameters to the module automatically in the following cases

- ▷ On restart
- ▷ When a module has been plugged into a configured slot (S7-400)
- ▷ Following the "return" of a rack or a distributed I/O station.

Static and dynamic module parameters

The module parameters are divided into static parameters and dynamic parameters. You can set both parameter types offline in the Hard-

ware Configuration. You can also modify the dynamic parameters at runtime by calling a system block. In the restart routine, the parameters set on the modules using system blocks are overwritten by the parameters set (and stored on the CPU) via the Hardware Configuration.

The parameters for the signal modules are in two data records: the static parameters in data record 0 and the dynamic parameters in data record 1. You can transfer both data records to the module with system function SFC 57 PARM_MOD, data record 0 or 1 with system function SFC 56 WR_DPARM, and only data record 1 with SFC 55 WR_PARM. The data records must be in the system data blocks on the CPU.

After parameterization of an S7-400 module, the specified values do not go into force until bit 2 ("Operating mode") in byte 2 of diagnostic data record 0 has assumed the value "RUN". The diagnostic data record 0 can be read with system function SFC 59 RD_REC or system function block SFB 52 RDREC.

Asynchronous processing of system blocks

Apart from the system function SFC 54 RD_DPARM, the system blocks for module parameterization and transmission of records work asynchronously. Execution of the function covers several calls, and is triggered by the block parameter REQ = "1". During processing of the job, the BUSY parameter is at "1", and the error information has the value W#16#7001 (job being processed). The error information is in the RET_VAL parameter for the system functions, and in bytes 2 and 3 of the STATUS parameter for the system function blocks.

A certain job for a module is specified by the module starting address and the data record number. As long as BUSY = "1", a renewed call for the same job with REQ = "1" has no effect, and the error information is set to W#16#7002.

If an error occurs when triggering an job, this is signaled by the error information, and BUSY remains "0".

If the job has been completed, BUSY has the status "0". If the job is completed without error, the error information has the value W#16#0000; with the system function SFC 59 RD_REC, the number of transmitted bytes is present in RET_VAL. In the event of an error, the error code is present in the error information.

Module and data record addressing

As far as addressing for data transfer is concerned, use the module start address. With mixed modules having input and output areas, use the lower area start address. If you assigned the same start address to both the input and output areas, use the identifier for an input address. Use the I/O identifier regardless of whether you want to execute a read or write operation.

The module start address is parameterized either using the IOID and LADDR parameters or – with newer system blocks – using the LADDR on its own. In this case, bit 15 determines whether an input ("0") or output ("1") is involved. (With the system function blocks SFB 52 RDREC and SFB 53 WRREC, this is the parameter ID.)

Enter the data record number in the RECNUM or INDEX parameter.

Use the RECORD parameter with the data type ANY to define an area of BYTE components. This may be a variable of type ARRAY, STRUCT or UDT, or an ANY pointer of type BYTE (for example P#DBzDBX*y.x* BYTE *.nnn*). If you use a variable, it must be a "complete" variable; individual array or structure components are not permissible.

Permissible data record numbers

Data records with the numbers 1 to 240 are permissible for the system functions for module parameterization. The specified data records must be present in the system data in the case of system blocks SFC 54 RD_DPARM, SFC 56 WR_DPARM and SFB 81 RD_DPAR.

System function SFC 58 WR_REC can process data records in the range 2 to 240, SFC 59 RD_REC in the range 0 to 240. System function blocks SFB 52 RDREC and SFB 53 WRREC transmit data records with numbers 0 to 255.

Data records 0 and 1 have a special significance with SIMATIC S7:

- Data record 0: reading of diagnostics data (4 bytes) and writing of static module parameters
- Data record 1: reading of diagnostics data (data record 0 and further data) and writing of dynamic module parameters

A data record can be up to 240 bytes long.

Module parameterization with PROFINET IO

Connection of distributed I/O over PROFINET IO requires an extended quantity framework for the module parameterization compared to PROFIBUS DP, and new system blocks are provided for this. These new system blocks can replace the previous blocks. Figure 22.3 provides an overview of the system blocks for module parameterization.

22.5.2 System Blocks for Module Parameterization

The following system blocks are available for parameterizing modules:

- SFB 81 RD_DPAR
  Read predefined parameters
- SFC 54 RD_DPARM
  Read predefined parameters
- SFC 55 WR_PARM
  Write dynamic parameters
- SFC 56 WR_DPARM
  Write predefined parameters
- SFC 57 PARM_MOD
  Parameterize module
- SFC 102 RD_DPARA
  Read predefined parameters

The parameters for the listed system blocks are described in Table 22.4.

SFB 81 RD_DPAR
Reading predefined parameters

System function block SFB 81 RD_DPAR transmits the data record with the number specified in the INDEX parameter from the corresponding SDB system data block to the destination area specified at the RECORD parameter.

The transmission is carried out in asynchronous mode, and can be divided between several program cycles; the BUSY parameter is "1" during the transmission. Following successful transmission, the VALID parameter is "1", and the LEN parameter contains the number of transmitted data bytes.

The read data record can now be evaluated or modified, for example, and written into the module using SFB 53 WRREC.

SFB 81 RD_DPAR replaces SFC 102 RD_DPARA and SFC 54 RD_PARM.

SFC 102 RD_DPARA
Reading predefined parameters

System function SFC 102 RD_DPARA transmits the data record with the number specified

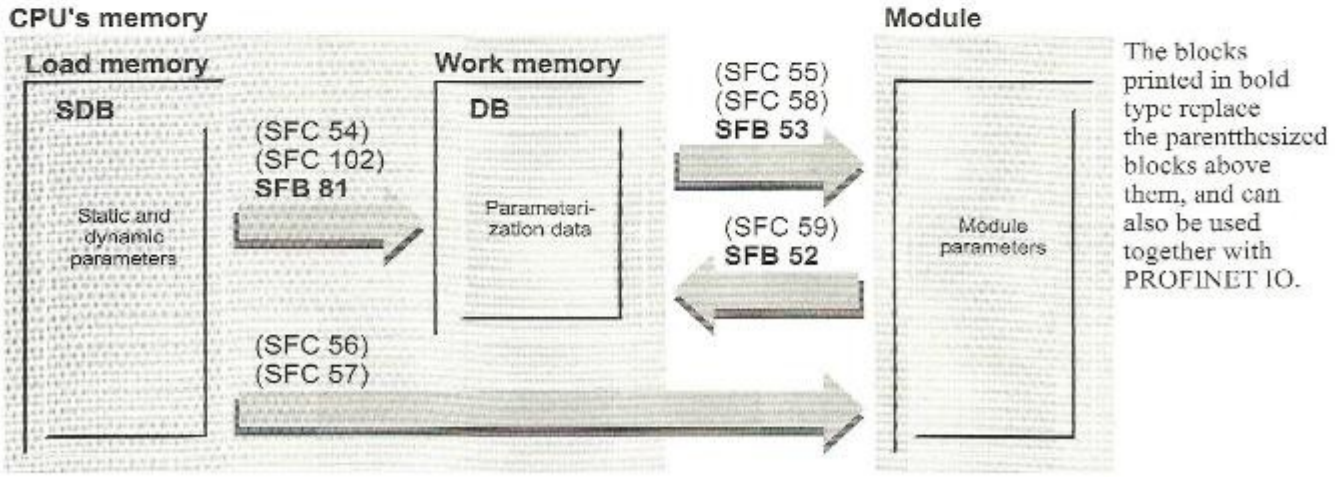


Figure 22.3 System Blocks for Module Parameterization

Table 22.4 Parameters of System Blocks for Module Parameterization

Present in SFC				Parameter	Declaration	Data Type	Contents, Description
-	55	56	57	REQ	INPUT	BOOL	"1" = write request
54	55	56	57	IOID	INPUT	BYTE	B#16#54 = peripheral inputs PI B#16#55 = peripheral outputs PQ
54	55	56	57	LADDR	INPUT	WORD	Module start address
54	55	56	-	RECNUM	INPUT	BYTE	Data record number
-	55	-	-	RECORD	INPUT	ANY	Source area for data record
54	55	56	57	RET_VAL	RETURN	INT	Error information
-	55	56	57	BUSY	OUTPUT	BOOL	Transfer still in progress if "1"
54	-	-	-	RECORD	OUTPUT	ANY	Destination area for data record

Present in		Parameter	Declaration	Data Type	Contents, Description
SFC 102	SFB 81	REQ	INPUT	BOOL	"1" = write request
SFC 102	SFB 81	LADDR	INPUT	WORD	Module start address
SFC 102	-	RECNUM	INPUT	BYTE	Data record number
-	SFB 81	INDEX	INPUT	INT	Data record number
SFC 102	-	RET_VAL	RETURN	INT	Error information
-	SFB 81	VALID	OUTPUT	BOOL	New data record received and valid
SFC 102	SFB 81	BUSY	OUTPUT	BOOL	Transfer still in progress if "1"
-	SFB 81	ERROR	OUTPUT	BOOL	Error has occurred if "1"
-	SFB 81	STATUS	OUTPUT	DWORD	Call identification or error information (bytes 2 and 3)
-	SFB 81	LEN	OUTPUT	INT	Length of read data
SFC 102	-	RECORD	OUTPUT	ANY	Destination area for data record
-	SFB 81	RECORD	IN_OUT	ANY	Destination area for data record

in the RECNUM parameter from the corresponding SDB system data block to the destination area specified at the RECORD.

The transmission is carried out in asynchronous mode, and can be divided between several program cycles; the BUSY parameter is "1" during the transmission.

SFC 102 replaces the SFC 54 RD_DPARM which works in synchronous mode.

SFC 54 RD_DPARM
Reading predefined parameters

System function SFC 54 RD_DPARM transfers the data record with the number specified in the RECNUM parameter from the relevant SDB system data block to the destination area specified at the RECORD parameter.

The transmission is carried out in asynchronous mode; the system function is processed until the data record has been transmitted. Since reading is from load memory, the relatively long processing time with large data records may – depending on the application – be considered as inconvenient. In this case use the SFB 81 RD_DPAR or the SFC 102 RD_DPARA which executes this function in asynchronous mode.

You can now, for example, evaluate or modify this read data record and write it to the module with SFB 53 WRREC or SFC 58 WR_REC.

SFC 55 WR_PARM
Writing dynamic parameters

System function SFC 55 WR_PARM transfers the data record addressed by RECORD to the

module specified by the IOID and LADDR parameters. Specify the number of the data record in the RECNUM parameter. The data record may only contain the dynamic module parameters, it must not be data record 0. If the module parameters are present in the associated SDB, they must not be identified as static.

When the job is initiated, the SFC reads the entire data record; the transfer may be distributed over several program scan cycles. The BUSY parameter is "1" during the transfer.

SFC 56 WR_DPARM
Writing predefined parameters

System function SFC 56 WR_DPARM transfers the data record with the number specified in the RECNUM parameter from the relevant SDB system data block to the module identified by the IOID and LADDR parameters.

The transfer may be distributed over several program scan cycles; the BUSY parameter is "1" during the transfer.

SFC 57 PARM_MOD
Parameterizing a module

System function SFC 57 PARM_MOD transfers all the data records programmed when the module was parameterized via the Hardware Configuration.

The transfer may be distributed over several program scan cycles; the BUSY parameter is "1" during the transfer.

22.5.3 Blocks for Transmitting Data Records

The following system blocks are available for transmitting data records:

- SFB 52 RDREC
  Read data record
- SFC 59 RD_REC
  Read data record
- SFB 53 WRREC
  Write data record
- SFC 58 WR_REC
  Write data record

The parameters of the listed system functions are described in Table 22.5 and those of the system function blocks in Table 22.6.

With an S7-300 CPU, you can process up to four write jobs and four read jobs simultaneously per DP segment. With an S7-400 CPU, up to eight write jobs and eight read jobs can be simultaneously active per DP segment. You can simultaneously execute a maximum total of 32 write jobs and 32 read jobs on external DP segments.

SFB 52 RDREC
Reading a data record

System function block SFB 52 RDREC with "1" at the REQ parameter reads the data record INDEX from the module and stores it in the destination area RECORD. The destination area must be longer than or at least as long as the data record. Use the MLEN parameter to specify how many bytes you wish to read.

Table 22.5 Parameters for System Functions Used for Data Transfer

Present in		Parameter	Declaration	Data Type	Contents, Description
SFC 58	SFC 59	REQ	INPUT	BOOL	"1" = write request
SFC 58	SFC 59	IOID	INPUT	BYTE	B#16#54 = input module B#16#55 = output module
SFC 58	SFC 59	LADDR	INPUT	WORD	Module start address
SFC 58	SFC 59	RECNUM	INPUT	BYTE	Data record number
SFC 58	-	RECORD	INPUT	ANY	Data record
SFC 58	SFC 59	RET_VAL	RETURN	INT	Error information
SFC 58	SFC 59	BUSY	OUTPUT	BOOL	Transfer still in progress if "1"
-	SFC 59	RECORD	OUTPUT	ANY	Data record

Table 22.6 Parameters for System Function Blocks Used for Data Transfer

Present in		Parameter	Declaration	Data Type	Contents, Description
SFB 52	SFB 53	REQ	INPUT	BOOL	"1" = write request
SFB 52	SFB 53	ID	INPUT	DWORD	Module start address Bit 15 = "0": Input address Bit 15 = "1": Output address
SFB 52	SFB 53	INDEX	INPUT	INT	Data record number
SFB 52	-	MLEN	INPUT	INT	Maximum number of bytes in data record to be read
-	SFB 53	LEN	INPUT	INT	Maximum number of bytes in data record to be sent
SFB 52	-	VALID	OUTPUT	BOOL	"1" = new data record has been received and is valid
-	SFB 53	DONE	OUTPUT	BOOL	Data record has been sent
SFB 52	SFB 53	BUSY	OUTPUT	BOOL	Transfer still in progress if "1"
SFB 52	SFB 53	ERROR	OUTPUT	BOOL	Error has occurred if "1"
SFB 52	SFB 53	STATUS	OUTPUT	DWORD	Status codes
SFB 52	-	LEN	OUTPUT	INT	Number of bytes of read data
SFB 52	SFB 53	RECORD	OUTPUT	ANY	Data record

The transfer may be distributed over several program scan cycles. The BUSY parameter is "1" during the transfer.

A signal status "1" at the VALID parameter signals that the data record has been read without faults. The LEN parameter then indicates the number of transmitted bytes.

In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

The system function block SFB 52 RDREC contains the functionality of system function SFC 59 RD_REC, and can replace the latter.

SFC 59 RD_REC
Reading a data record

When the REQ parameter is "1", SFC 59 RD_REC reads the data record addressed by the RECNUM parameter from the module and places it in destination area RECORD. The destination area must be longer than or at least as long as the data record. If the transfer is completed without error, the RET_VAL parameter contains the number of bytes transferred.

The transfer may be distributed over several program scan cycles; the BUSY parameter is "1" during the transfer.

S7-300 delivered prior to February 1997: the SFC reads as much data from the specified data record as the destination area can accommodate. The size of the destination area may not exceed that of the data record.

SFB 53 WRREC
Writing a data record

System function block SFB 53 WRREC with "1" at the REQ parameter writes the data record INDEX from the source area RECORD to the module. Use the LEN parameter to specify how many bytes you wish to write.

The transfer may be distributed over several program scan cycles. The BUSY parameter is "1" during the transfer.

A signal status "1" at the DONE parameter signals that the data record has been written without faults. In the event of an error, the ERROR parameter is set to "1" and the error information output in the STATUS parameter.

The system function block SFB 53 WRREC contains the functionality of system function SFC 58 WR_REC, and can replace the latter.

SFC 58 WR_REC
Writing a data record

SFC 58 WR_REC transfers the data record addressed by the RECORD parameter and the number RECNUM to the module defined by the IOID and LADDR parameters. A "1" in the REQ parameter starts the transfer. When the job is initiated, the SFC reads the complete data record.

The transfer may be distributed over several program scan cycles; the BUSY parameter is "1" during the transfer.

23 Error Handling

The CPU reports errors or faults detected by the modules or by the CPU itself in different ways:

- Errors in arithmetic operations (overflow, invalid REAL number) by setting status bits (status bit OV, for example, for a numerical overflow)
- Errors detected while executing the user program (synchronous errors) by calling organization blocks OB 121 and OB 122
- Errors in the programmable controller which do not relate to program scanning (asynchronous errors) by calling organization blocks OB 80 to OB 87

The CPU signals the occurrence of an error or fault, and in some cases the cause, by setting error LEDs on the front panel. In the case of unrecoverable errors (such as invalid OP code), the CPU goes directly to STOP.

With the CPU in STOP mode, you can use a programming device and the CPU information functions to read out the contents of the block stack (B stack), the interrupt stack (I stack) and the local data stack (L stack) and then draw conclusions as to the cause of error.

The system diagnostics can detect errors/faults on the modules, and enters these errors in a diagnostic buffer. Information on CPU mode transitions (such as the reasons for a STOP) are also placed in the diagnostic buffer.

The contents of this buffer are retained on STOP, on a memory reset, and on power failure, and can be read out following power recovery and execution of a start-up routine using a programming device.

On the new CPUs, you can use CPU parameterization to set the number of entries the diagnostics buffer is to hold.

23.1 Synchronous Errors

The CPU's operating system generates a synchronous error when an error occurs in immediate conjunction with program scanning. A distinction is made between two error types:

A **programming error** is present if execution of the program is faulty. Such errors include BCD conversion errors, errors with indirect addressing; addressing of missing timers, counters or blocks. Organization block OB 121 is called in the event of a programming error.

An **access error** (PZF) is present if an attempt is made to access a faulty or non-existent module or an I/O address not known on the CPU. The operating system reacts in different manners depending on the type of access:

- The I/O access is from the user program. In this case, the I/O access error organization block OB 122 is called.
- The PZF occurs during automatic updating of a (partial) process image. With S7-300 CPUs the default response is that an entry is not made into the diagnostics buffer and an OB is not called; S7-400 CPUs enter each PZF into the diagnostics buffer and start the OB 85. With newer CPUs, the response to a PZF can be parameterized (see "Program execution errors OB 85" in Chapter 23.3, "Asynchronous Errors").
- The PZF occurs if a partial process image is updated by a system function. The error and the address of the first byte signaling the error are then returned in their parameters (system functions SFC 26 UPDAT_PI, SFC 27 UPDAT_PO, SFC 126 SYNC_PI and SFC 127 SYNC_PO).

If the corresponding organization block OB 121 or OB 122 is not present when a synchronous error event occurs, the CPU enters the STOP status.

Table 23.1 Start Information for the Synchronous Error OBs 121 and 122

Byte	Variable Name	Data Type	Description, Contents
0	OB12x_EV_CLASS	BYTE	B#16#25 = Call programming error OB 121 B#16#29 = Call access error OB 122
1	OB12x_SW_FLT	BYTE	Error code (see Chapter 23.2.1, "Error Filters")
2	OB12x_PRIORITY	BYTE	Priority class in which the error occurred
3	OB12x_OB_NUMBR	BYTE	OB number (B#16#79 or B#16#80)
4	OB12x_BLK_TYPE	BYTE	Type of block interrupted (S7-400 only) OB: B#16#88, DB: B#16#8A, FB: B#16#8E, FC: B#16#8C
5	OB121_RESERVED_1 OB122_MEM_AREA	BYTE	Byte assignments (B#16#xy): 7... (x) ... 4: 1 Bit access; 2 Byte access; 3 Word access; 4 Doubleword access 3 ... (y) ... 0: 0 I/O area PI or PQ; 1 Process-image input table I; 2 Process-image output table Q; 3 Memory bits M; 4 Global data block DB; 5 Instance data block DI; 6 Temporary local data L; 7 Temporary local data of the predecessor block V
6...7	OB121_FLT_REG OB122_MEM_ADDR	WORD	OB 121: Error source: ▷ Errored address (at read/write access) ▷ Errored area (in the case of area error) ▷ Incorrect number of the block, timer/counter function OB 122: Address at which the error occurred
8...9	OB12x_BLK_NUM	WORD	Number of the block in which the error occurred (S7-400 only)
10...11	OB12x_PRG_ADDR	WORD	Error address in the block that caused the error (S7-400 only)
12...19	OB12x_DATE_TIME	DT	Time at which programming error was detected

Table 23.1 shows the start information for both synchronous error organization blocks.

The S7-400 CPUs distinguish between two types of access error: access to a non-existent module and invalid access attempt to an existing module (acknowledgment delay QVZ). If a module fails during operation, that module is marked "non-existent" approximately 150 μs after an access attempt, and an I/O access error (PZF) is reported on every subsequent attempt to access the module. The CPU also reports an I/O access error when an attempt is made to access a non-existent module, regardless of whether the attempt was direct (via the I/O area) or indirect (via the process image).

If an access error occurs when writing the peripheral outputs, an S7-400 CPU tracks the output process image, an S7-300 CPU does not.

A synchronous error OB has the same priority (class) as the block in which the error occurred. The accumulators and address registers of asynchronous error OB contain the values present in the block causing the error at the time of aborting. The data block registers are deleted; the condition code word has an undefined content.

Note that when a synchronous error OB is called, its 20 bytes of start information are also pushed onto the L stack for the priority class that caused the error, as are the other temporary local data for the synchronous error OB and for all blocks called in this OB. The area reserved for the temporary local data must be designed for this in every associated priority class (program execution level); (fixed setting with S7-300 CPUs, adjustable with S7-400 CPUs during

Table 23.2 SFC Parameters for Synchronous Error Handling

SFC	Parameter	Declaration	Data Type	Contents, Description
36	PRGFLT_SET_MASK	INPUT	DWORD	New (additional) programming error filter
	ACCFLT_SET_MASK	INPUT	DWORD	New (additional) access error filter
	RET_VAL	RETURN	INT	W#16#0001 = The new filter overlaps the existing filter
	PRGFLT_MASKED	OUTPUT	DWORD	Complete programming error filter
	ACCFLT_MASKED	OUTPUT	DWORD	Complete access error filter
37	PRGFLT_RESET_MASK	INPUT	DWORD	Programming error filter to be reset
	ACCFLT_RESET_MASK	INPUT	DWORD	Access error filter to be reset
	RET_VAL	RETURN	INT	W#16#0001 = The new filter contains bits that are not set (in the current filter)
	PRGFLT_MASKED	OUTPUT	DWORD	Remaining programming error filter
	ACCFLT_MASKED	OUTPUT	DWORD	Remaining access error filter
38	PRGFLT_QUERY	INPUT	DWORD	Programming error filter to be queried
	ACCFLT_QUERY	INPUT	DWORD	Access error filter to be queried
	RET_VAL	RETURN	INT	W#16#0001 = The query filter contains bits that are not set (in the current filter)
	PRGFLT_CLR	OUTPUT	DWORD	Programming error filter with error messages
	ACCFLT_CLR	OUTPUT	DWORD	Access error filter with error messages

parameterization of the CPU in the "Memory" tab).

It is similar with the block nesting depth. The nesting depth per priority class permissible for a CPU is the total of the nesting depth of the "normal" processing and the nesting depth of the synchronous error processing.

In the case of S7-400, another synchronous error OB can be called in an error OB. The block nesting depth for a synchronous error OB is 3 for S7-400 CPUs and 4 for S7-300 CPUs.

You can disable and enable a synchronous error OB call with system functions SFC 36 MSK_FLT, SFC 37 DMSK_FLT and SFC 38 READ_ERR.

23.2 Synchronous Error Handling

The following system functions are provided for handling synchronous errors:

- SFC 36 MSK_FLT
  Mask synchronous errors
  (disable OB call)
- SFC 37 DMSK_FLT
  Unmask synchronous error
  (re-enable OB call)
- SFC 38 READ_ERR
  Read error register

The operating system enters the synchronous error in the diagnostic buffer without regard to the use of system functions SFC 36 to SFC 38. The parameters for these system functions are listed in Table 23.2.

23.2.1 Error Filters

The error filters are used to control the system functions for synchronous error handling. In the programming error filter, one bit stands for each programming error detected; in the access error filter, one bit stands for each access error detected. When you define an error filter, you set the bit that stands for the synchronous error you want to mask, unmask or query. The error filters returned by the system functions show a "1" for synchronous errors that are still masked or which have occurred.

Table 23.3 Access Error Filter

Bit	Error Code	Contents
3	B#16#42	I/O access error on read S7-300 and CPU 417: Module does not exist or does not acknowledge S7-400 (except CPU 417): An existing module does not acknowledge after first I/O access (time-out)
4	B#16#43	I/O access error on write S7-300 and CPU 417: Module does not exist or does not acknowledge S7-400 (except CPU 417): An existing module does not acknowledge after first I/O access (time-out)
5	B#16#44	S7-400 only (except CPU 417): I/O access error on attempt to read non-existent modules (PZF) or (in the case of older revision levels) repeated access to modules which do not acknowledge
6	B#16#45	S7-400 only (except CPU 417): I/O access error on attempt to write to non-existent modules (PZF) or (in the case of older revision levels) repeated access to modules which do not acknowledge

The access error filter is shown in Table 23.3; the Error Code column shows the contents of variable OB122_SW_FLT in the start information for OB 122.

The programming error filter is shown in Table 23.4; the Error Code column shows the contents of variable OB121_SW_FLT in the start information for OB 121.

The error filter bits not listed in the tables are not relevant to the handling of synchronous errors.

Table 23.4 Programming Error Filter

Bit	Error Code	Contents
1	B#16#21	BCD conversion error (pseudo-tetrad detected during conversion)
2	B#16#22	Area length error on read (address not within area limits)
3	B#16#23	Area length error on write (address not within area limits)
4	B#16#24	Area length error on read (wrong area in area pointer)
5	B#16#25	Area length error on write (wrong area in area pointer)
6	B#16#26	Invalid timer number
7	B#16#27	Invalid counter number
8	B#16#28	Address error on read (bit address <>0 in conjunction with byte, word or doubleword access and indirect addressing)
9	B#16#29	Address area on write (bit address <>0 in conjunction with byte, word or doubleword access and indirect addressing)
16	B#16#30	Write error, global data block (write-protected block)
17	B#16#31	Write error, instance data block (write-protected block)
18	B#16#32	Invalid number of a global data block (DB register)
19	B#16#33	Invalid number of an instance data block (DI register)
20	B#16#34	Invalid number of a function (FC)
21	B#16#35	Invalid number of a function block (FB)
26	B#16#3A	Called data block (DB) does not exist
28	B#16#3C	Called function (FC) does not exist
30	B#16#3E	Called function block (FB) does not exist

23.2.2 Masking Synchronous Errors

System function SFC 36 MSK_FLT disables synchronous error OB calls via the error filters. A "1" in the error filters indicates the synchronous errors for which the OBs are not to be called (the synchronous errors are "masked"). The masking of synchronous errors in the error filters is in addition to the masking stored in the operating system's memory. SFC 36 returns a function value indicating whether a (stored) masking already exists on at least one bit for the masking specified at the input parameters (W#16#0001).

SFC 36 returns a "1" in the output parameters for all currently masked errors.

If a masked synchronous error event occurs, the respective OB is not called and the error is entered in the error register. The Disable applies to the current priority class (priority level). For example, if you were to disable a synchronous error OB call in the main program, the synchronous error OB would still be called if the error were to occur in an interrupt service routine.

23.2.3 Unmasking Synchronous Errors

System function SFC 37 DMSK_FLT enables the synchronous error OB calls via the error filters. You enter a "1" in the filters to indicate the synchronous errors for which the OBs are once again to be called (the synchronous errors are "unmasked"). The entries corresponding to the specified bits are deleted in the error register. SFC 37 returns W#16#0001 as function value if no (stored) masking already exists on at least one bit for the unmasking specified at the input parameters.

SFC 37 returns a "1" in the output parameters for all currently masked errors.

If an unmasked synchronous error occurs, the respective OB is called and the event entered in the error register. The Enable applies to the current priority class (priority level).

23.2.4 Reading the Error Register

System function SFC 38 READ_ERR reads the error register. You must enter a "1" in the error filters to indicate the synchronous errors whose entries you want to read. SFC 38 returns W#16#0001 as function value when the selection specified in the input parameters included at least one bit for which no (stored) masking exists.

SFC 38 returns a "1" in the output parameters for the selected errors when these errors occurred, and deletes these errors in the error register when they are queried. The synchronous errors that are reported are those in the current priority class (priority level).

23.2.5 Entering a Substitute Value

SFC 44 REPL_VAL allows you to enter a substitute value in accumulator 1 from within a synchronous error OB. Use SFC 44 when you can no longer read any values from a module (for instance when a module is defective). When you program SFC 44, OB 122 ("access error") is called every time an attempt is made to access the module in question. When you call SFC 44, you can load a substitute value into the accumulator; the program scan is then resumed with the substitute value. Table 23.5 lists the parameters for SFC 44.

You may call SFC 44 in only one synchronous error OB (OB 121 or OB 122).

Table 23.5 Parameters for SFC 44 REPL_VAL

SFC	Parameter Name	Declaration	Data Type	Contents, Description
44	VAL	INPUT	DWORD	Substitute value
	RET_VAL	RETURN	INT	Error information

23.3 Asynchronous Errors

Asynchronous errors are errors which can occur independently of the program scan. When an asynchronous error occurs, the operating system calls one of the organization blocks listed below:

OB 80 Timing error

OB 81 Power supply error

OB 82 Diagnostic interrupt

OB 83 Insert/remove module interrupt

OB 84 CPU hardware fault

OB 85 Program execution error

OB 86 Rack failure

OB 87 Communication error

OB 88 Processing abort

The OB 82 call (diagnostic interrupt) is described in detail in Chapter 23.4, "System Diagnostics".

On the S7-400H, there are three additional asynchronous error OBs:

OB 70 I/O redundancy errors

OB 72 CPU redundancy errors

OB 73 Communications redundancy errors

The call of these asynchronous error organization blocks can be disabled and enabled with system functions SFC 39 DIS_IRT and SFC 40 EN_IRT, and delayed and enabled with system functions SFC 41 DIS_AIRT and SFC 42 EN_AIRT.

Timing errors OB 80

The operating system calls organization block OB 80 when one of the following errors occurs:

- ▷ Cycle monitoring time exceeded
- ▷ OB request error (the requested OB is still executing or an OB was requested too frequently within a given priority class)
- ▷ Time-of-day interrupt error (TOD interrupt time past because clock was set forward or after transition to RUN)

If no OB 80 is available and a timing error occurs, the CPU goes to STOP. The CPU also goes to STOP if the OB is called a second time in the same program scan cycle.

Power supply errors OB 81

The operating system calls organization block OB 81 if one of the following errors occurs:

- ▷ At least one backup battery in the central controller or in an expansion unit is empty
- ▷ No battery voltage in the central controller or in an expansion unit
- ▷ 24 V supply failed in central controller or in an expansion unit

OB 81 is called for incoming and outgoing events. If there is no OB 81, the CPU continues functioning when a power supply error occurs.

Insert/remove module interrupt OB 83

The operating system monitors the module configuration once per second. An entry is made in the diagnostic buffer and in the system status list each time a module is inserted or removed in RUN, STOP or START-UP mode.

In addition, the operating system calls organization block OB 83 if the CPU is in RUN mode. If there is no OB 83, the CPU goes to STOP on an insert/remove module interrupt.

As much as a second can pass before the insert/remove module interrupt is generated. As a result, it is possible that an access error or an error relating to the updating of the process image could be reported in the interim between removal of a module and generation of the interrupt.

If a suitable module is inserted into a configured slot, the CPU automatically parameterizes that module, using data records already stored on that CPU. Only then is OB 83 called in order to signal that the connected module is ready for operation.

CPU hardware faults OB 84

The operating system calls organization block OB 84 when an interface error (MPI network, PROFIBUS DP) occurs or disappears. If there is no OB 84, CPUs with older operating systems go to STOP on a CPU hardware fault.

Program execution errors OB 85

The operating system calls organization block OB 85 when one of the following errors occurs:

- ▷ Start request for an organization block which has not been loaded
- ▷ Error occurred while the operating system was accessing a block (for instance no instance data block when a system function block (SFB) was called)
- ▷ I/O access error while executing (automatic) updating of the process image on the system side

On the S7-400 CPUs, OB 85 is called at an I/O access error on the system side, i.e. when updating the process image in each cycle. The substitute value or zero is then entered in the relevant byte in the process-image input table at every update.

On the S7-300 CPUs, OB 85 is not called in the event of an I/O access error during automatic updating of the process image. At the first errored access, the substitute value or zero is entered in the relevant byte; it is then no longer updated.

With appropriately equipped CPUs, you can use CPU parameterization to influence the call mode of OB 85 in the event of an I/O access error on the system side:

- ▷ OB 85 is called every time. The affected input byte is overwritten with the substitute value or with zero each time.
- ▷ OB 85 is called in the event of the first error with the attribute "incoming". An affected input byte is only overwritten with the substitute value or with zero the first time; following this it is no longer updated. If the error is then corrected, OB 85 is called with the attribute "outgoing"; following this, it is updated "normally".
- ▷ OB 85 is not called in the event of an access error. Affected input bytes are overwritten once with the substitute value or zero, and then no longer updated.

If there is no OB 85, the CPU goes to STOP on a program execution error.

Rack failure OB 86

The operating system calls organization block OB 86 if it detects the failure of a rack (power failure, line break, defective IM; not with S7-300), a DP master system or a distributed I/O station (PROFIBUS DP or PROFINET IO). OB 86 is called for both incoming and leaving errors.

In multiprocessor mode, OB 86 is called in all CPUs if a rack fails.

If there is no OB 86, the CPU goes to STOP if a rack failure occurs.

Communication error OB 87

The operating system calls organization block OB 87 when a communication error occurs. Some examples of communication errors are:

- ▷ Invalid frame identification or frame length detected during global data communication
- ▷ Sending of diagnostic entries not possible
- ▷ Clock synchronization error
- ▷ GD status cannot be entered in a data block

If there is no OB 87, the CPU goes to STOP when a communication error occurs.

Processing abort OB 88

The operating system calls organization block OB 88 when processing of a block in the user program is aborted. Possible reasons for an abort include:

- ▷ With a synchronous error, the permissible block nesting depth is exceeded.
- ▷ With a block call, the permissible nesting depth is exceeded.
- ▷ A fault has occurred when allocating the local data of a block.

If there is no OB 88, the CPU goes to STOP if a processing abort occurs. The CPU also goes to STOP if the OB is called in priority class 28.

I/O redundancy error OB 70

The operating system of an H CPU calls organization block OB 70 if a redundancy loss occurs on PROFIBUS DP, e.g. in the event of a

bus failure on the active DP master or in the event of a fault in the interface of a DP slave.

If OB 70 does not exist, the CPU continues to operate in the event of an I/O redundancy error.

CPU redundancy error OB 72

The operating system of an H CPU calls organization block OB 72 if one of the following events occurs:

- Redundancy loss of the CPU
- Comparison error (e.g. in RAM, in the PIQ)
- Standby-master changeover
- Synchronization error
- Error in a SYNC submodule
- Update abort

If OB 72 does not exist, the CPU continues to operate in the event of a CPU redundancy error.

Communications redundancy error OB 73

The operating system of a fault-tolerant CPU calls the organization block OB 73 when the redundancy of a fault-tolerant S7 connection is lost for the first time. If at least one fault-tolerant S7 connection signals a loss of redundancy, the OB 73 is not called again when there is a further loss of redundancy.

If OB 73 does not exist, the CPU continues to operate in the event of a communications redundancy error.

23.4 System Diagnostics

23.4.1 Diagnostic Events and Diagnostic Buffer

System diagnostics is the detection, evaluation and reporting of errors occurring in programmable controllers. Examples are errors in the user program, module failures or wirebreaks on signaling modules. These *diagnostic events* may be:

- Diagnostic interrupts from modules with this capability
- System errors and CPU mode transitions
- User messages via system functions.

Modules with diagnostic capabilities distinguish between programmable and non-programmable diagnostic events. Programmable diagnostic events are reported only when you have set the parameters necessary to enable diagnostics. Non-programmable diagnostic events are always reported, regardless of whether or not diagnostics have been enabled. In the event of a reportable diagnostic event,

- The fault LED on the CPU goes on
- The diagnostic event is passed on to the CPU's operating system
- A diagnostic interrupt is generated if you have set the parameters enabling such interrupts (by default, diagnostic interrupts are disabled).

All diagnostic events reported to the CPU operating system are entered in a *diagnostic buffer* in the order in which they occurred, and with date and time stamp. The diagnostic buffer is a battery-backed memory area on the CPU which retains its contents even in the event of a memory reset. The diagnostic buffer is a ring buffer whose size depends on the CPU. When the diagnostic buffer is full, the oldest entry is overwritten by the newest.

You can read out the diagnostic buffer with a programming device at any time. In the CPU's *System Diagnostics* parameter block you can specify whether you want expanded diagnostic entries (all OB calls). You may also specify whether the last diagnostic entry made before the CPU goes to STOP should be sent to a specific node on the MPI bus.

23.4.2 Writing User Entries in the Diagnostic Buffer

System function **SFC 52 WR_USMSG** writes an entry in the diagnostic buffer which may be sent to all nodes on the MPI bus Table 23.6 lists the parameters for SFC 52.

The entry in the diagnostic buffer corresponds in format to that of a system event, for instance the start information for an organization block. Within the permissible boundaries, you may choose your own event ID (EVENTN parameter) and additional information (INFO1 and INFO2 parameters).

Table 23.6 Parameters for SFC 52 WR_USMSG

SFC	Parameter Name	Declaration	Data Type	Contents, Description
52	SEND	INPUT	BOOL	If "1": Send is enabling
	EVENTN	INPUT	WORD	Event ID
	INFO1	INPUT	ANY	Additional information 1 (one word)
	INFO2	INPUT	ANY	Additional information 2 (one doubleword)
	RET_VAL	RETURN	INT	Error information

The event ID is identical to the first two bytes of the buffer entry (Figure 23.1). Permissible for a user entry are the event classes 8 (diagnostic entries for signal modules), 9 (standard user events), A and B (arbitrary user events).

Additional information (INFO1) corresponds to bytes 7 and 8 of the buffer entry (one word) and additional information 2 (INFO2) to bytes 9 to 12 (one doubleword). The contents of both variables may be of the user's own choice.

Set SEND to "1" to send the diagnostic entry to the relevant node. Even if sending is not possible (because no node is logged in or because the Send buffer is full, for example), the entry is still made in the diagnostic buffer (when bit 9 of the event ID is set).

23.4.3 Evaluating Diagnostic Interrupts

When a diagnostic interrupt is incoming or outgoing, the operating system interrupts scanning of the user program and calls **organization block OB 82**. If OB 82 has not been programmed, the CPU goes to STOP on a diagnostic interrupt. You can disable or enable OB 82 with system function SFC 39 DIS_IRT or SFC 40 EN_IRT, and delay or enable it with system function SFC 41 DIS_AIRT or SFC 42 EN_AIRT.

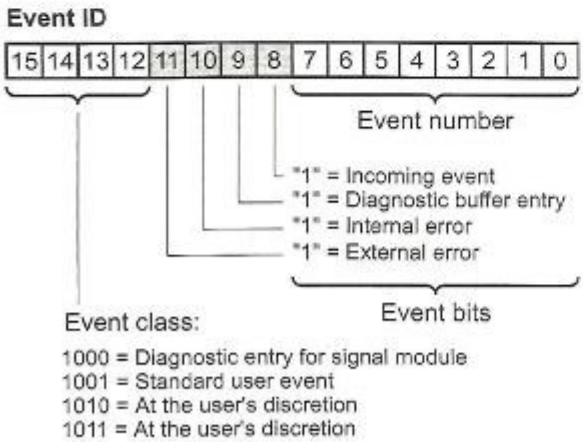


Figure 23.1 Event ID for Diagnostic Buffer Entries

PROFIBUS DPV1 slaves can also generate a diagnostic interrupt if the master CPU is at STOP. A diagnostic interrupt triggered with the CPU at STOP is acknowledged, but not processed. Calling the organization block OB 82 is not carried out subsequently when the CPU goes to RUN.

Table 23.7 shows the start information of the diagnostic interrupt OB 82. In the first byte of the start information, B#16#39 stands for an incoming diagnostic interrupt and B#16#38 for a leaving diagnostic interrupt. The sixth byte gives the address identifier (B#16#54 stands for an input, B#16#55 for an output); the subsequent INT variable contains the address of the module that generated the diagnostic interrupt. The next four bytes contain the diagnostic information provided by that module.

You can use system function SFC 59 RD_REC (read data record) in OB 82 to obtain detailed error information. The diagnostic information are consistent until OB 82 is exited, that is, they remain "frozen". Exiting of OB 82 acknowledges the diagnostic interrupt on the module.

A module's diagnostic data are in data records DS 0 and DS 1. Data record DS 0 contains four bytes of diagnostic data describing the current status of the module. The contents of these four bytes are identical to the contents of bytes 8 to 11 of the OB 82 start information. Data record DS 1 contains the four bytes from data record DS 0 and, in addition, the module-specific diagnostic data.

Table 23.7 Start Information of Organization Block OB 82 (Diagnostics Interrupt)

Byte	Variable Name	Data Type	Contents, Description
0	OB82_EV_CLASS	BYTE	B#16#38 = DOWN event B#16#39 = UP event
1	OB82_FLT_ID	BYTE	Error code (B#16#42)
2	OB82_PRIORITY	BYTE	Priority class for the diagnostic interrupt OB
3	OB82_OB_NUMBR	BYTE	OB number (B#16#52)
4	OB82_RESERVED_1	BYTE	Reserved
5	OB82_IO_FLAG	BYTE	I/O identification (B#16#54 = input, B#16#55 = output)
6...7	OB82_MDL_ADDR	WORD	Module starting address of module sending the interrupt
8.0	OB82_MDL_DEFECT	BOOL	Module fault
8.1	OB82_INT_FAULT	BOOL	Internal fault
8.2	OB82_EXT_FAULT	BOOL	External fault
8.3	OB82_PNT_INFO	BOOL	Channel fault present
8.4	OB82_EXT-VOLTAGE	BOOL	External power supply missing
8.5	OB82_FLD_CONNCTR	BOOL	Front connector missing
8.6	OB82_NO_CONFIG	BOOL	Module not parameterized
8.7	OB82_CONFIG_ERR	BOOL	Incorrect parameters in module
9	OB82_MDL_TYPE	BYTE	Bits 0 to 3: module class Bit 4: channel information present Bit 5: user information present Bit 6: diagnostics interrupt from proxy Bit 7: reserved
10.0	OB82_SUB_MDL_ERR	BOOL	User module wrong or missing
10.1	OB82_COMM_FAULT	BOOL	Communications fault
10.2	OB82_MDL_STOP	BOOL	Operating status ("0" = RUN, "1" = STOP)
10.3	OB82_WTCH_DOG_FLT	BOOL	Timeout has triggered
10.4	OB82_INT_PS_FLT	BOOL	Module-internal power supply failed
10.5	OB82_PRIM_BATT_FLT	BOOL	Flat battery
10.6	OB82_BCKUP_BATT_FLT	BOOL	Complete battery backup failed
10.7	OB82_RESERVED_2	BOOL	Reserved
11.0	OB82_RACK_FLT	BOOL	Expansion unit failed
11.1	OB82_PROC_FLT	BOOL	Processor failure
11.2	OB82_EPROM_FLT	BOOL	EPROM fault
11.3	OB82_RAM_FLT	BOOL	RAM fault
11.4	OB82_ADU_FLT	BOOL	ADC/DAC fault
11.5	OB82_FUSE_FLT	BOOL	Blown fuse
11.6	OB82_HW_INTR_FLT	BOOL	Hardware interrupt lost
11.7	OB82_RESERVED_3	BOOL	Reserved
12...19	OB82_DATE_TIME	DT	Acquisition time of diagnostics event

When using a CPU with DPV1 capability and a corresponding slave, you can use the system function block SFB 54 RALRM to obtain further information on the diagnostic interrupt.

23.4.4 Reading the System Status List

The system status list (SSL) describes the current status of the programmable controller. Using information functions, the list can be read but not modified. Since the complete system status list is extremely large, reading is carried out in sublists and sublist extracts. Sublists are virtual lists, which means that they are made available by the CPU operating system only on request.

The SSL ID exists in order to identify a sublist. This contains the module type class to which the list applies, the number of the sublist extract, and the actual SSL sublist number (Figure 23.2). You are provided with the desired information together with the index which specifies an object of a sublist. As standard, the CPU provides information on the automation system, but FM and CP modules can also use this service in order to make information available (see module documentation). The possible system status lists for a CPU are described in the operations description.

Reading the header information

Use the SSL ID W#16#0Fxx to read the header information of an SSL sublist without the associated data record (xx = SSL sublist number). The SSL_HEADER. N_DR parameter (number of data records) then returns the maximum possible number of data records of the sublist extract which the module can deliver with an

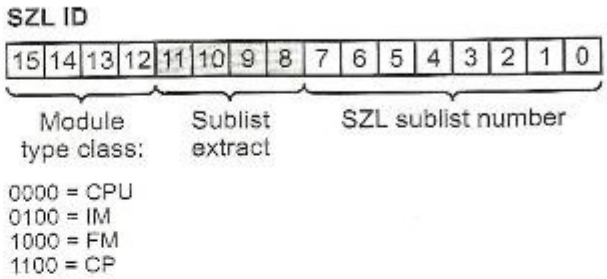


Figure 23.2 Structure of SSL ID

SSL job. With dynamic sublists, the value can be greater than the actual number which can be read. The length of a data record is specified in SSL_HEADER. With this data in the header information, it is possible e.g. to create a sufficiently large data buffer for the associated SSL sublist in the startup.

SFC 51 RDSYSST
Read SSL sublist

Use system function SFC 51 RDSYSST to read a sublist or a sublist extract of the system status list (SSL). The parameters for SFC 51 are listed in Table 23.8.

REQ = "1" initiates the read operation, and BUSY = "0" tells you when it has been completed. The operating system can execute several asynchronous read operations quasi simultaneously; how many depends on the CPU being used. If SFC 51 reports a lack of resources via the function value (W#16#8085), you must resubmit your read request.

The contents of parameters SSL_ID and INDEX are CPU-dependent. If the INDEX parameter is not required to provide information, its assignment is irrelevant. The SSL_

Table 23.8 Parameters for SFC 51 RDSYSST

SFC	Parameter	Declaration	Data Type	Contents, Description
51	REQ	INPUT	BOOL	If "1": Submit request
	SSL_ID	INPUT	WORD	Sublist ID
	INDEX	INPUT	WORD	Type or number of the sublist object
	RET_VAL	RETURN	INT	Error information
	BUSY	OUTPUT	BOOL	If "1": Read not yet completed
	SSL_HEADER	OUTPUT	STRUCT	Length and number of data records read
	DR	OUTPUT	ANY	Field for data records read

HEADER parameter is of data type STRUCT, with variables LENGTHDR (data type WORD) and N_DR (WORD) as components. LENGTHDR contains the length of a data record, N_DR the number of data records read.

Use the DR parameter to specify the variable or data area in which SFC 51 is to enter the data records. For example, P#DB200.DBX0.0 WORD 256 would provide an area of 256 data words in data block DB 200, beginning with DBB 0. If the area provided is of insufficient capacity, as many data records as possible will be entered. Only complete data records are transferred. The specified area must be able to accommodate at least one data record.

23.5 Web Server

CPUs with an Ethernet interface can have a Web server that provides information from the CPU. To read out the information via the company's intranet or the Internet, you require a Web browser, such as Internet Explorer, Version 6.0 or higher, that displays the information on HTML pages.

23.5.1 Activating the Web server

You can activate the Web server when parameterizing the CPU with the Hardware Configuration. When a CPU is selected, EDIT → OBJECT PROPERTIES selects the "Web" tab on which you activate the Web server and select the language for the message texts and the entries in the diagnostics buffer. It is also possible to select several languages, depending on the memory capacity of the CPU.

The languages installed with STEP 7 are available. You can establish agreement with the languages installed in the project in the SIMATIC Manager: Select OPTIONS → LANGUAGE FOR DISPLAY DEVICES and define the languages. The Web server is ready for use after the configuration data have been loaded onto the CPU.

23.5.2 Reading out Web information

To dial the CPU in the Web browser, enter the IP address of the CPU in the "Address" field in the form *http://aaa.bbb.ccc.ddd*. You can find the IP address of the CPU in the Object Properties of the PROFINET interface on the "General" tab.

The Web server also supports the Terminal Service of Windows so that thin-client solutions can be used with mobile devices or HMI stations with the thin-client option under Windows CE. In this case, specify the address in the form *http://aaa.bbb.ccc.ddd/basic*.

You can navigate to further information from the start page of the CPU. Please note that the information offered is static and you have to provide for updating of the screen content yourself. The current information from the CPU is used for printing out, regardless of what is on the display.

Note: Protect your Web server against unauthorized access by means of a firewall.

23.5.3 Web information

The Web server can provide the following information in a suitably equipped CPU:

- ▷ Start page with general CPU information
- ▷ Identification data
- ▷ Diagnostics buffer
- ▷ Messages (without acknowledge facility)
- ▷ PROFINET interface
- ▷ Variable status
- ▷ Variable tables

The first page provided by the Web server is the Welcome page. From here, click on ENTER to reach the Start page. If you want to skip this intro page in future, activate the option "Skip Intro".

Start and identification

The Start page shows you general information and the status of the CPU at the time of the query. The Identification page contains the key data of the CPU, such as the plant designation, location designation, and order number.

Diagnostics buffer

You see the contents of the diagnostics buffer on this page. Select the number of diagnostics

buffer entries per display interval. Detailed information about the selected event is displayed.

You can select the display language in the top right window. If the selected language is not configured, the information is displayed in hexadecimal code.

Messages

Messages are displayed in chronological order, including the date and time. You cannot acknowledge the messages via the Web browser.

You can search for specific information with filter settings. With sort functions, you can sort the messages according to message number or status. Detailed information about the selected message is displayed.

You can select the display language in the top right window. If the selected language is not configured, the information is displayed in hexadecimal code.

PROFINET interface

Information on the PROFINET interface is found on the pages "Parameters" and "Statistics". The MAC address and the IP address are displayed, for example, as are statistical analyses of sent and received data packages.

Variable status

You can monitor the status of up to 50 variables on this page. When you specify the address of the variables and the display format, you receive the value of the variable.

You can select the display language in the window at the top right. When specifying addresses, please note that the mnemonics for English (e.g. I for input) differs from those of other languages (E for "Eingang" in German, for example). Syntax errors are indicated in red.

Variable tables

The Web server allows you to monitor up to 50 variable tables with up to 200 variables each. The memory space available in the CPU might not be sufficient to make use of all the possibilities. If variable tables are displayed incomplete, reduce the memory required by the messages and symbol comments, if possible, use only one language, and keep the number of variables per table low.

Select one of the configured variable tables for display. First, you must prepare the variable table for use by the Web server. If you select EDIT → OBJECT PROPERTIES when a variable table is selected, or if you create a new variable table, the Properties window opens. Enter *VAToWEB* as the family on the "General – Part 2" tab, or alternatively, activate the "Web server" checkbox.

Appendix

This section of the book contains useful supplements to the LAD and FBD programming languages, an overview of the contents of the STEP 7 Block Libraries, and a function overview of all LAD and FBD elements.

- You can also provide blocks with LAD/FBD program with **block protection.** For this purpose, you use the source-oriented Editor in the STL programming language.
- You can use a further function of STL, **indirect addressing**, to transfer data areas in the LAD and FBD programming languages; the addresses of these data areas are then not calculated until runtime. The "LAD_Book" and "FBD_Book" libraries which you can download from the publisher's Website (see page 8) each contain a "Sample Message Frame" showing how to set up and transfer data areas.
- The standard STEP 7 package includes **block libraries** with loadable functions and function blocks and with block headers and interface descriptions for system blocks (SFCs and SFBs).
- A **function overview** of all LAD and FBD functions completes the book.

You can download the archive libraries "LAD_Book" and "FBD_Book" from the publisher's Website (see page 8). You retrieve these libraries under the SIMATIC Manager with FILE → RETRIEVE. Select the archive from the dialog field displayed. You define the destination directory in the next dialog field. In general, libraries are located under ...\STEP7\S7LIBS; but you can choose any other directory, for example ...\STEP7\S7PROJ, which normally contains the projects.

The "LAD_Book" and "FBD_BOOK" libraries each contain eight programs which are essentially illustrative examples of LAD and FBD representations. Two extensive examples show the programming of functions, function blocks and local instances (Conveyor Example) and the handling of data (Message Frame Example). The memory requirements are approximately 2 MB.

To try out an example, set up a project that corresponds to your hardware configuration and copy the program, including the symbol table, from the library to the project. Now you can test the example online.

24 Supplements to Graphic Programming
Block Protection; Indirect Addressing; Message Frame Example

25 Block Libraries
Organization Blocks, System Function Blocks, IEC Function Blocks, S5-S7 Converting Blocks, TI-S7 Converting Blocks, PID Control Blocks, Communication Blocks

26 LAD Function Overview
All LAD functions

27 FBD Function Overview
All FBD functions

24 Supplements to Graphic Programming

24.1 Block Protection

The keyword KNOW_HOW_PROTECT represents block protection. You cannot view, print or modify a block with this attribute. The Editor only displays the block header and the declaration table with the block parameters. In source-oriented input, you can protect any block yourself with KNOW_HOW_PROTECT. This means that no one, including yourself, can view the compiled block (keep the source file in a safe place!).

You can enter the KNOW_HOW_PROTECT block protection with STL, and it must be source-oriented. To do so, proceed as follows:

1) Create the block under LAD or FBD in the usual way. Later, this block will be overwritten in the user program *Blocks* by the block with the keyword. If you want to retain the (original) block (strongly recommended when entering block protection), you can store the block in, for example, a (user-created) library before entering the keyword. You can also store your entire user program in this way.

2) Create a source container. If there is no object *Sources* under the *S7 program* (on the same level as the user program *Blocks),* you must create one: Select the *S7 program* and insert the object *Sources* with INSERT → S7 SOFTWARE → SOURCE FOLDER.

3) Generate an STL source from the block. Change to the Editor (via the taskbar, for example, or open any block in *Blocks* and then close it again) and select the menu item FILE → GENERATE SOURCE. In the dialog form displayed, set your project, select the object *Sources,* and assign a name for the source file under "Object Name". Confirm with "OK". The next dialog form shows you all blocks in the *Blocks* container; select the block(s) from which you want to create a source file. Confirm with "OK".

4) Open the source file (for example by double-clicking on the source file symbol in the SIMATIC Manager or with the Editor and FILE → OPEN). You now see the ASCII source of your LAD/FBD block. If you have previously selected several blocks, these blocks will be arranged in order in the source file.

The entries for a code block are in the following order:

- Keyword for the block type (FUNCTION, FUNCTION_BLOCK, ORGANIZATION_BLOCK) with specification of the address. This may be followed by the block title (starting with TITLE=...) and the block comment (starting with //...).
- Block attributes (depending on whether you have filled in fields in the Properties page for the block and, if so, how many).
- Variable declaration (several sections with the keywords VAR_xxx, ..., END_VAR); depending on whether you have declared block-local variables and, if so, which ones.
- Program, starts with BEGIN and ends with the keyword for the end of the block (for example END_FUNCTION_BLOCK).

The entries for a data block are in the following order:

- Keyword for the block type (DATA_BLOCK) with specification of the address.
- Block attributes (depending on whether you have filled in fields in the Properties page for the block and, if so, how many).
- Variable declaration (starting with STRUCT and ending with END_STRUCT) or, in the case of instance data blocks, the address of the associated function block.

- ▷ Variable initialization, starting with BEGIN and ending with END_DATA_BLOCK.

5) You enter the keyword KNOW_HOW_PROTECT in the source file, in its own line in each case, following the block attributes and before the variable declaration. If you have created a source from several blocks, enter the keywords in all selected blocks. Finally, store the source file.

6) Compile the source file with FILE → COMPILE. The compiler creates a block with the specified block attributes (in the creation language STL in the case of code blocks; the creation language is not significant here since KNOW_HOW_PROTECT means the block can no longer be viewed or printed out). The compiled (new) block is in user program *Blocks* and replaces the (old) block with the same number.

You can set the program editor using OPTIONS → CUSTOMIZE on the "Sources" tab so that it automatically generates a source when saving a block. You can use the "Run" button to generate new sources from existing blocks.

24.2 Indirect Addressing

With the programming language STL, you have a method of accessing operands whose addresses are not calculated until runtime. This is also possible to a limited degree in LAD or FBD: You can wait until runtime to define which data areas you want to copy with SFC 20 BLKMOV.

However, first some useful information on pointers.

24.2.1 Pointers: General Remarks

For indirect addressing, you require a data format that contains the bit address as well as the byte address and, if applicable, the operand area. This data format is a pointer. A pointer is also used to point to an operand.

There are three types of pointers:

- ▷ Area pointers; these are 32 bits long and contain a specific operand or its address
- ▷ DB pointers; these are 48 bits long and in addition to the area pointer they also contain the number of the data block
- ▷ ANY pointers; these are 80 bits long and, in addition to the DB pointer, they contain further information such as the data type of the operand

24.2.2 Area Pointer

The area pointer contains the operand address and, if applicable, the operand area. Without operand area, it is an area-internal pointer; if the pointer also contains an operand area, it is referred to as an area-crossing pointer.

The notation for constant representation is as follows:

P#y.x	for an area-internal pointer e.g. P#22.0
P#Zy.x	for an area-crossing pointer e.g. P#M22.0

where x = bit address, y = byte address and Z = area. You specify the operand ID as the area. The contents of bit 31 differentiates the two pointer types.

Figure 24.1 shows all pointer types and their contents as provided by STEP 7.

The area pointer has, in principle, a bit address that must always be specified even with digital operands; in the case of digital operands, specify 0 as the bit address. Example: You can use area pointer P#M22.0 to address memory bit M 22.0, but also memory byte MB 22, memory word MW 22 or memory doubleword MD 22.

24.2.3 DB Pointer

In addition to the area pointer, a DB pointer also contains a data block number in the form of a positive INT number. It specifies the data block if the area pointer points to the global data or instance data area. In all other cases, the first two bytes contain zero.

The notation of the pointer is familiar to you from full addressing of data operands. Here also, the data block and the data operand are

Area-internal pointer

Byte n	Byte n+1	Byte n+2	Byte n+3
0 0 0 0 0 0 0 0	0 0 0 0 0 y y y	y y y y y y y y	y y y y y x x x
	Byte address		Bit address

Area-crossing pointer

Byte n	Byte n+1	Byte n+2	Byte n+3
1 0 0 0 0 Z Z Z	0 0 0 0 0 y y y	y y y y y y y y	y y y y y x x x
Operand area	Byte address		Bit address

DB pointer

Byte n	Data block
Byte n+1	number
Byte n+2	
Byte n+3	Area
Byte n+4	pointer
Byte n+5	

	ANY pointer for data types	**ANY pointer** for timers/counters	**ANY pointer** for blocks
Byte n	16#10	16#10	16#10
Byte n+1	Type	Type	Type
Byte n+2	Quantity	Quantity	Quantity
Byte n+3			
Byte n+4	Data block	16#0000	16#0000
Byte n+5	number		
Byte n+6		Type	16#0000
Byte n+7	Area	16#00	
Byte n+8	pointer	Number	Number
Byte n+9			

Address area:

0	0	0	Peripheral I/O (P)
0	0	1	Inputs (I)
0	1	0	Outputs (Q)
0	1	1	Memory bits (M)
1	0	0	Global data (DBX)
1	0	1	Instance data (DIX)
1	1	0	Temporary local data (L)[1]
1	1	1	Temporary local data of the predecessor block (V)[2]

[1] Not with area-crossing addressing

[2] Only with block parameter transfer

Type in the ANY pointer:

Elementary data types		**Complex data types**
01	BOOL	0E DT
02	BYTE	13 STRING
03	CHAR	**Parameter types**
04	WORD	17 BLOCK_FB
05	INT	18 BLOCK_FC
06	DWORD	19 BLOCK_DB
07	DINT	1A BLOCK_SDB
08	REAL	1C COUNTER
09	DATE	1D TIMER
0A	TOD	**Zero pointer**
0B	TIME	00 NIL
0C	S5TIME	

Figure 24.1 Structure of the Pointers in STEP 7

specified, separated by a period: P#DataBlock. DataOperand

Example: P#DB 10.DBX 20.5

You can apply this pointer to a block parameter of parameter type POINTER in order to point to a data operand. The Editor uses this pointer type internally to transfer actual parameters.

24.2.4 ANY Pointer

In addition to the DB pointer, the ANY pointer also contains the data type and a repetition factor. This makes it possible to point to a data area

The ANY pointer is available in two variants: For variables with data types and for variables with parameter types. If you point to a variable with a data type, the ANY pointer contains a DB pointer, the type, and a repetition factor. If the ANY pointer points to a variable with a parameter type, it contains only the number instead of the DB pointer in addition to the type. In the case of a timer or counter function, the type is repeated in byte (n+6); byte (n+7) contains B#16#00. In all other cases, these two bytes contain the value W#16#0000.

The first byte of the ANY pointer contains the syntax ID; in STEP 7 it is always 10_{hex}. The type specifies the data type of the variables for which the ANY pointer applies. Variables of elementary data type, DT and STRING receive the type shown in Figure 24.1 and the quantity 1.

If you apply a variable of data type ARRAY or STRUCT (also UDT) at an ANY parameter, the Editor generates an ANY pointer to the field or the structure. This ANY pointer contains the ID for BYTE (02_{hex}) as the type, and the byte length of the variable as the quantity. The data type of the individual field or structure components is not significant here. Thus, an ANY pointer points with double the number of bytes to a WORD field. Exception: A pointer to a field consisting of components of data type CHAR is also created with CHAR type (03_{hex}).

You can apply an ANY pointer at a block parameter of parameter type ANY if you want to point to a variable or an operand area. The constant representation for data types is as follows:
P#[DataBlock.]Operand Type Quantity

Examples:

- ▷ P#DB 11.DBX 30.0 INT 12
  Area with 12 words in DB 11 beginning DBB 30
- ▷ P#M 16.0 BYTE 8
  Area with 8 bytes beginning MB 16
- ▷ P#E 18.0 WORD 1
  Input word IW 18
- ▷ P#E 1.0 BOOL 1
  Input I 1.0

In the case of parameter types, you write the pointer as follows: L#Number Type Quantity

Examples:

- ▷ L#10 TIMER 1
  Timer function T 10
- ▷ L#2 COUNTER 1
  Counter function C 2

The Editor then applies an ANY pointer that agrees in type and quantity with the specifications in the constant representation. Please note that the operand address in the ANY pointer must also be a bit address for data types.

Specification of a constant ANY pointer makes sense if you want to access a data area for which you have not declared any variables. In principle, you can also apply variables or operands at an ANY parameter. For example, the representation "P#I 1.0 BOOL 1" is identical with "I 1.0" or the corresponding symbolic address.

If you do not specify any defaults when declaring an ANY parameter at a function block, the Editor assigns 10_{hex} to the syntax ID and 00_{hex} to the remaining bytes. The Editor then represents this (empty) ANY pointer (in the data view) as follows: P#P0.0 VOID 0.

24.2.5 "Variable" ANY Pointer

When copying with SFC 20, you specify in the source and destination parameters either an absolute-addressed area (for example, P#DB127.DBX0.0 BYTE 32) or a variable. In both cases, the source area and the destination area are fixed during programming (variable indexing is not possible even for array components). The following method is available for runtime modification of a data area created at a block parameter of type ANY:

You create a variable of data type ANY in the temporary local data and use this to initialize an ANY parameter. The Program Editor then does not generate an ANY pointer (as it would if you created a different variable), but uses the ANY variable in the temporary local data as an ANY pointer to a source or destination area. The ANY variable in the temporary local data is structured in the same way as an ANY pointer; you can now modify the individual entries at runtime.

This procedure functions not only in the case of SFC 20 BLKMOV but also with block parameters of type ANY in other blocks.

The "LAD_BOOK" and "FBD_Book" libraries which you can download from the publisher's Website (see page 8), each contain a "Message Frame Example" program, which in turn both contain an example of the "variable" ANY pointer.

24.3 Brief Description of the "Message Frame Example"

This example deals primarily with how data is handled, and is broken down as follows:

- ▷ Message frame data, shows how to handle data structures
- ▷ Time-of-day check, shows how to handle system blocks and standard blocks
- ▷ Editing the message frame, shows the use of SFC 20 BLKMOV with fixed addresses
- ▷ Indirect copying of the data area, shows an "indirect copy" function using "variable" ANY pointers
- ▷ Save message frame, shows the use of "indirect copying"

Figure 24.2 shows the program and data structure for this example.

You can find this program under "Message Frame Example" in the libraries "LAD_Book" and FBD_BOOK" which you can download from the publisher's Website (see page 8).

Message frame data

The example shows how you can define frequently occurring data structures as your own data type and how you use this data type when declaring variables and parameters.

We construct a data store for incoming and outgoing message frames: A Send mailbox with the structure of a message frame, a Receive mailbox with the same structure, and a (receive) ring buffer that is to provide intermediate storage for incoming message frames. Since the data structure of the message frame occurs frequently, we will define it as a user-defined data type (UDT) frame. The *message frame* contains a frame header whose structure we also want to give a name to. The Send mailbox and the Receive mailbox are to be data blocks that each contain a variable with the structure of the *frame*. Finally, there is also a ring buffer, a data block with an array consisting of eight components that also have the same data structure as the *frame*.

Time-of-day check

The example shows how to handle system and standard blocks (evaluating errors, copying from the library, renaming).

The time-of-day check function is to output the time-of-day in the integral CPU real-time clock as a function value. For this purpose, we require the system function SFC 1 READ_CLK, which reads the date and the time-of-day from the real-time clock in the DATE_AND_TIME or DT data format. Since we only want to read the time-of-day, we also require the IEC function FC 8 DT_TOD. This function fetches the time-of-day in the TIME_OF_DAY or TOD format from the DT data format.

Error evaluation

The system functions signal an error via binary result parameter BR and via function value RET_VAL. An error occurred if binary result BR = "0"; the function value is then also negative (bit 15 is set). The IEC standard functions signal an error only via the binary result. Both types of error evaluation are shown in the example. If an error was encountered, an invalid value is output for the time-of-day. In addition, the binary result is affected. After the

Message frame example

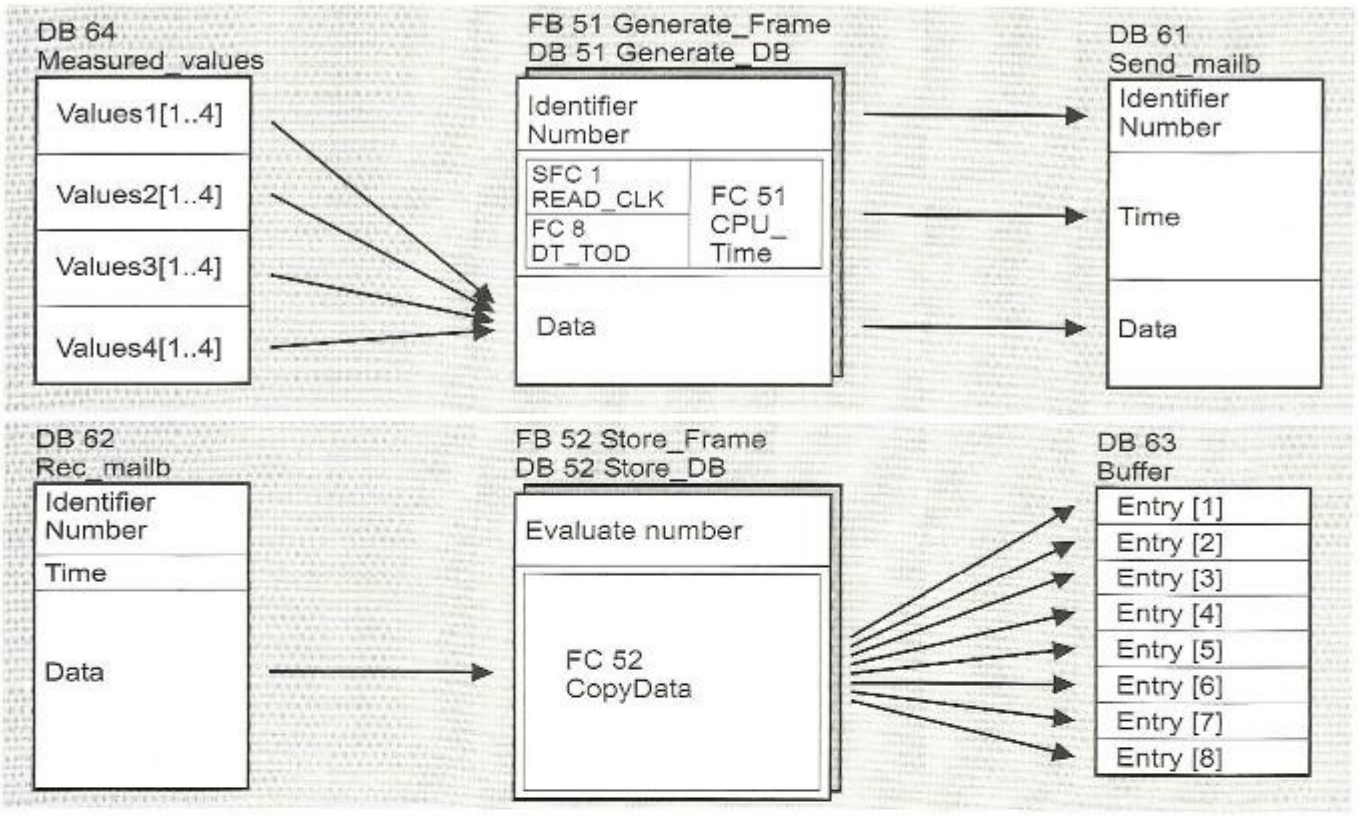


Figure 24.2 Data Structure for the Message Frame Data Example

time-of-day check function has been called, you can therefore also use the binary result to see if an error has occurred.

Offline programming of system functions

Before saving the input block, system function SFC 1 and standard function FC 8 must be included in the offline user program. Both functions are included in the STEP 7 standard package. You will find these functions in the block libraries provided. (For the system functions integrated into the CPU, the library contains only an interface description, not the actual system functions program. The function can be called offline via this interface description; the interface description is not transferred to the CPU. Loadable functions such as the IEC functions are available in the library as executable programs.)

Select the *Standard Library* with FILE → OPEN under the SIMATIC Manager and open the *System Function Blocks* library. Under *Blocks, you* will find all interface descriptions for the system functions. If you still have the project window of your project open, you can display the two windows side-by-side with WINDOW → ARRANGE → VERTICALLY and "drag" the selected system functions into your program using the mouse (select the SFC with the mouse, hold the mouse key down, drag to *Blocks* or into its open window, and "drop"). Copy standard function FC 8 in the same way. You will find it on the *IEC Function Blocks* library. FC 8 is a loadable function; it therefore reserves user memory, in contrast to SFC 1.

If a standard function block is called under "Libraries" in the Program Element Catalog using the Editor, it is automatically copied into *Blocks* and entered in the Symbol Table.

Renaming standard functions

You can rename a loadable standard function. Select the standard function (for example FC 8) in the project window and click once (again) on

the designator. The name appears in a frame and you can specify a new address (for example, FC 98). If you press the F1 key while the standard function (renamed to FC 98) is still selected, you will nevertheless receive the online help for the original standard function (FC 8).

If an identically addressed block exists when copying, a dialog box appears where you can choose between overwriting and renaming.

Symbolic address

You can assign names to the system functions and the standard functions in the Symbol Table so that you can also access these functions symbolically. You can assign these names freely within the permissible limits applicable to block names. In the example, the block name in each case is selected as a symbolic name (for better identification).

Editing the message frame

The data block *Send_Mailb is* to be filled with the data for a message frame. We use a function block that has the ID and the consecutive number stored in its instance data block. The net data are stored finally in a global data block; they are copied into the Send mailbox with system function BLKMOV. We use the time-of-day check function to take the time-of-day from the CPU's real-time clock.

The first network in the function block FB *Generate_Frame* transfers the ID stored in the instance data block to the frame header. The consecutive number is incremented by +1 and is also transferred to the frame header.

The second network contains the *READ_CLK* function call that takes the time-of-day from the real-time clock and enters it in the frame header in TIME_OF_DAY format.

In the subsequent networks, you will see a method of copying selected variables at runtime with the system function SFC 20 BLKMOV and without using indirect addressing. It is therefore also not necessary to know the absolute address or the structure of the variables. The principle is extremely simple: The desired copy function is selected using comparison functions. The numbers 1 to 4 are permissible as selection criteria.

FB *Generate_Frame is* programmed in such a way that it is called via a signal edge to generate a message frame.

Indirect copying of a data area

The example shows the editing and use of a "variable" ANY pointer with graphical program elements.

The *CopyData* function copies a data area whose address and length you can set as required via block parameters. The individual block parameters correspond to the individual elements of an ANY pointer (see Chapter 24.2.4, "ANY Pointer"). The specifications in the block parameters must be valid values; they are not checked (SFC 20 BLKMOV signals a copy error in its function value parameter, which is transferred to the *CopyData* function's function value parameter).

Essential elements are the two temporary variables *SoPointer* and *DesPointer*, which are *of* data type ANY. They contain the ANY pointers for the system function SFC 20 BLKMOV. *SoPointer* points to the source area of the data to be transferred and *DesPointer* points to the destination area. Figure 24.3 shows the structure of the *SoPointer* variable; *DesPointer* has the same structure. The individual bytes, words and doublewords of the ANY variables are accessed via their absolute addresses.

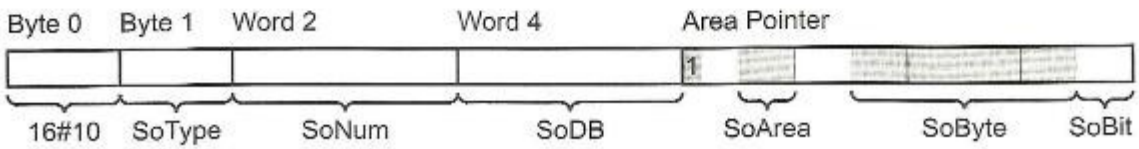


Figure 24.3 Structure of the Variable *SoPointer*

Save message frame

The example shows the use of the *CopyData* function (copying a data area with programmable address).

A message frame in the data block *Rec_Mailb is* to be written to the next location in data block *Buffer*. The block-local variable *Entry* determines the location in the ring buffer; the value of this location is used to calculate the address in the ring buffer.

The *Entry* variable has a value range of from 0 to 7. In the first network, a comparator determines whether *Entry is* less than 7. If this is the case, *Entry is* incremented by 1 in the next network, otherwise it is set to zero. *Entry* multiplied by 16 gives the absolute byte address of the next entry in the ring buffer (the data structure *Message frame* consists of 16 bytes).

The *CopyData* function, which copies the message frame from the Receive mailbox (data block DB 62) to the ring buffer (data block DB 63), is called in Network 3.

25 Block Libraries

The STEP 7 Basic software includes the **Standard Library** which contains the following library programs:

- ▷ Organization Blocks
- ▷ System Function Blocks
- ▷ IEC Function Blocks
- ▷ S5-S7 Converting Blocks
- ▷ TI-S7 Converting Blocks
- ▷ PID Control Blocks
- ▷ Communication Blocks
  Communication functions
- ▷ Miscellaneous Blocks
  Time synchronization and stamping

Further supplied libraries are **SIMATIC_NET_CP,** which contains the communications blocks for the CP modules in the library programs *CP 300* and *CP 400*, **Redundant IO (V1)** with blocks for module redundancy, and **Redundant IO CGP** with blocks for the redundancy of individual module channels.

You can copy blocks or interface descriptions into own projects from the library programs described.

25.1 Organization Blocks

(Prio = Default priority class)

OB	Prio	Designation
1	1	Main program
10	2	Time-of-day interrupt 0
11	2	Time-of-day interrupt 1
12	2	Time-of-day interrupt 2
13	2	Time-of-day interrupt 3
14	2	Time-of-day interrupt 4
15	2	Time-of-day interrupt 5
16	2	Time-of-day interrupt 6
17	2	Time-of-day interrupt 7

OB	Prio	Designation
20	3	Time-delay interrupt 0
21	4	Time-delay interrupt 1
22	5	Time-delay interrupt 2
23	6	Time-delay interrupt 3
30	7	Watchdog interrupt 0 (5 s)
31	8	Watchdog interrupt 1 (2 s)
32	9	Watchdog interrupt 2 (1 s)
33	10	Watchdog interrupt 3 (500 ms)
34	11	Watchdog interrupt 4 (200 ms)
35	12	Watchdog interrupt 5 (100 ms)
36	13	Watchdog interrupt 6 (50 ms)
37	14	Watchdog interrupt 7 (20 ms)
38	15	Watchdog interrupt 8 (10 ms)
40	16	Hardware interrupt 0
41	17	Hardware interrupt 1
42	18	Hardware interrupt 2
43	19	Hardware interrupt 3
44	20	Hardware interrupt 4
45	21	Hardware interrupt 5
46	22	Hardware interrupt 6
47	23	Hardware interrupt 7
55	2	DPV1 status interrupt
56	2	DPV1 update interrupt
57	2	DPV1 vendor interrupt
60	25	Multiprocessor interrupt

OB	Prio	Designation
61	25	Synchronous cycle interrupt 0
62	25	Synchronous cycle interrupt 1
63	25	Synchronous cycle interrupt 2
64	25	Synchronous cycle interrupt 3
65	25	Technology synchronous interrupt
70	25	I/O redundancy error
72	28	CPU redundancy error
73	25	Communication redundancy error
80	26	Time error [1]
81	26	Power supply fault [1]
82	26	Diagnostics interrupt [1]
83	26	Insert/remove-module interrupt [1]
84	26	CPU hardware fault [1]
85	26	Priority class error [1]
86	26	DP error [1]
87	26	Communications error [1]
88	28	Processing abort
90	29	Background processing
100	27	Warm restart
101	27	Hot restart
102	27	Cold restart
121	-	Programming error
122	-	I/O access error

[1] Prio = 28 at restart

25.2 System Function Blocks

CPU clock and run-time meter

SFC	Name	Designation
0	SET_CLK	Set clock
1	READ_CLK	Read clock
2	SET_RTM	Set run-time meter
3	CTRL_RTM	Modify run-time meter
4	READ_RTM	Read run-time meter
48	SNC_RTCB	Synchronize slave clocks
64	TIME_TCK	Read system time
100	SET_CLKS	Set time and clock status
101	RTM	Use run-time meter

IEC timers and IEC counters

SFB	Name	Designation
0	CTU	Up counter
1	CTD	Down counter
2	CTUD	Up/down counter
3	TP	Pulse
4	TON	On delay
5	TOF	Off delay

S7 communication

SFB	Name	Designation
8	USEND	Uncoordinated send
9	URVC	Uncoordinated receive
12	BSEND	Block-oriented send
13	BRCV	Block-oriented receive
14	GET	Read data from partner
15	PUT	Write data to partner
16	PRINT	Write data to printer
19	START	Initiate cold or warm start in the partner
20	STOP	Set partner to STOP
21	RESUME	Initiate restart in the partner
22	STATUS	Check status of partner
23	USTATUS	Receive status of partner

SFC	Name	Designation
62	CONTROL	Check communications status
87	C_DIAG	Determine connection status

S7 basic communication

SFC	Name	Designation
65	X_SEND	Send data externally
66	X_RCV	Receive data externally
67	X_GET	Read data externally
68	X_PUT	Write data externally
69	X_ABORT	Abort external connection
72	I_GET	Read data internally
73	I_PUT	Write data internally
74	I_ABORT	Abort internal connection

Global data communications

SFC	Name	Designation
60	GD_SND	Send GD packet
61	GD_RCV	Receive GD packet

S7-300C point-to-point coupling

SFB	Name	Designation
60	SEND_PTP	Send data (ASCII, 3964 (R))
61	RCV_PTP	Receive data (ASCII, 3964(R))
62	RES_RCVB	Delete receive buffer (ASCII, 3964(R))
63	SEND_RK	Send data (RK 512)
64	FETCH_RK	Fetch data (RK 512)
65	SERVE_RK	Receive and provide data (RK 512)

Integral S7-300C functions

SFB	Name	Designation
44	ANALOG	Positioning with analog output
46	DIGITAL	Positioning with digital output
47	COUNT	Control counter
48	FREQUENC	Control frequency measurement
49	PULSE	Control pulse-width modulation

Integrated functions CPU 312/314/614

SFB	Name	Designation
29	HS_COUNT	High-speed counter
30	FREQ_MES	Frequency meter
38	HSC_A_B	Control "Counter A/B"
39	POS	Control "Positioning"
41	CONT_C	Continuous closed-loop control
42	CONT_S	Step-action control
43	PULSEGEN	Generate pulse

SFC	Name	Designation
63	AB_CALL	Call assembler block

Drum

SFB	Name	Designation
32	DRUM	Drum

H-CPU

SFC	Name	Designation
90	H_CTRL	Control operating modes on H-CPU

Interrupt events

SFC	Name	Designation
28	SET_TINT	Set time-of-day interrupt
29	CAN_TINT	Cancel time-of-day interrupt
30	ACT_TINT	Activate time-of-day interrupt
31	QRY_TINT	Query time-of-day interrupt
32	SRT_DINT	Start time-delay interrupt
33	CAN_DINT	Cancel time-delay interrupt
34	QRY_DINT	Query time-delay interrupt
35	MP_ALM	Trigger multiprocessor alarm
36	MSK_FLT	Mask synchronous errors
37	DMSK_FLT	Unmask synchronous errors
38	READ_ERR	Read event status register
39	DIS_IRT	Disable interrupt events
40	EN_IRT	Enable interrupt events
41	DIS_AIRT	Delay interrupt events
42	EN_AIRT	Enable interrupt events

Address modules

SFC	Name	Designation
5	GADR_LGC	Determine logical address
49	LGC_GADR	Determine slot
50	RD_LGADR	Determine all logical addresses
70	GEO_LOG	Determine logical address
71	LOG_GEO	Determine slot

Data record transfer

SFB	Name	Designation
52	RDREC	Read data record from a DP slave
53	WRREC	Write data record to a DP slave
81	RD_PAR	Read predefined parameters

SFC	Name	Designation
54	RD_DPARM	Read predefined parameter
55	WR_PARM	Write dynamic parameter
56	WR_DPARM	Write predefined parameter
57	PARM_MOD	Parameterize module
58	WR_REC	Write data record
59	RD_REC	Read data record
102	RD_DPARA	Read predefined parameter

Process image updating

SFC	Name	Designation
26	UPDAT_PI	Update process-image input table
27	UPDAT_PO	Update process-image output table
79	SET	Set I/O bit field
80	RSET	Reset I/O bit field
126	SYNC_PI	Update partial process image of inputs in isochrone mode
127	SYNC_PO	Update partial process image of outputs in isochrone mode

System diagnostics

SFC	Name	Designation
6	RD_SINFO	Read start information
51	RDSYSST	Read SYS ST sublist
52	WR_USMSG	Entry in the diagnostics buffer

Distributed I/O

SFB	Name	Designation
54	RALRM	Receive interrupt
75	SALRM	Trigger interrupt

SFC	Name	Designation
7	DP_PRAL	Initiate hardware interrupt
11	DPSYN_FR	SYNC/FREEZE
12	D_ACT_DP	Deactivate or activate DP slave
13	DPNRM_DG	Read diagnostics data
14	DPRD_DAT	Read slave data
15	DPWR_DAT	Write slave data
103	DP_TOPOL	Determine bus topology

Create block-related messages

SFB	Name	Designation
31	NOTIFY_8P	Messages without acknowledgment display
33	ALARM	Messages with acknowledgment display
34	ALARM_8	Messages without accompanying values
35	ALARM_8P	Messages with accompanying values
36	NOTIFY	Messages without acknowledgment display
37	AR_SEND	Send archive data

SFC	Name	Designation
9	EN_MSG	Enable messages
10	DIS_MSG	Disable messages
17	ALARM_SQ	Messages that can be acknowledged
18	ALARM_S	Messages that are always acknowledged
19	ALARM_SC	Determine acknowledgment status
105	READ_SI	Read dynamic system resources
106	DEL_SI	Enable dynamic system resources

107	ALARM_DQ	Messages that can be acknowledged
108	ALARM_D	Messages that are always acknowledged

Copy and block functions

SFC	Name	Designation
20	BLKMOV	Copy data area
21	FILL	Pre-assign data area
22	CREAT_DB	Create data block in work memory
23	DEL_DB	Delete data block
24	TEST_DB	Test data block
25	COMPRESS	Compress memory
44	REPL_VAL	Enter substitute value
81	UBLKMOV	Copy data area without gaps
82	CREA_DBL	Generate data block in load memory
83	READ_DBL	Read load memory
84	WRIT_DBL	Write load memory
85	CREA_DB	Create data block in work memory

Program control

SFC	Name	Designation
43	RE_TRIGR	Retrigger cycle time monitor
46	STP	Change to STOP state
47	WAIT	Wait for delay time
78	OB_RT	Determine OB runtime
104	CIR	Configuration in RUN
109	PROTECT	Changing program protection

Blocks for PROFINET CbA

SFC	Name	Designation
112	PN_IN	Update inputs
113	PN_OUT	Update outputs
114	PN_DP	Update DP connections

25.3 IEC Function Blocks

String functions

FC	Name	Designation
21	LEN	Length of a STRING
20	LEFT	Left section of a STRING
32	RIGHT	Right section of a STRING
26	MID	Middle section of a STRING
2	CONCAT	Concatenate STRINGs
17	INSERT	Insert STRING
4	DELETE	Delete STRING
31	REPLACE	Replace STRING
11	FIND	Find STRING
16	I_STRNG	Convert INT to STRING
5	DI_STRNG	Convert DINT to STRING
30	R_STRNG	Convert REAL to STRING
38	STRNG_I	Convert STRING to INT
37	STRNG_DI	Convert STRING to DINT
39	STRNG_R	Convert STRING to REAL

Date and time functions

FC	Name	Designation
3	D_TOD_DT	Combine DATE and TOD to DT
6	DT_DATE	Extract DATE from DT
7	DT_DAY	Extract day-of-the-week from DT
8	DT_TOD	Extract TOD from DT
33	S5TI_TIM	Convert S5TIME to TIME
40	TIM_S5TI	Convert TIME to S5TIME
1	AD_DT_TM	Add TIME to DT
35	SB_DT_TM	Subtract TIME from DT
34	SB_DT_DT	Subtract DT from DT

Comparisons

FC	Name	Designation
9	EQ_DT	Compare DT for equal to
28	NE_DT	Compare DT for not equal to
14	GT_DT	Compare DT for greater than
12	GE_DT	Compare DT for greater than or equal to

23	LT_DT	Compare DT for less than
18	LE_DT	Compare DT for less than or equal to
10	EQ_STRNG	Compare STRING for equal to
29	NE_STRNG	Compare STRING for not equal to
15	GT_STRNG	Compare STRING for greater than
13	GE_STRNG	Compare STRING for greater than or equal to
24	LT_STRNG	Compare STRING for less than
19	LE_STRNG	Compare STRING for less than or equal to

Math functions

FC	Name	Designation
22	LIMIT	Limiter
25	MAX	Maximum selection
27	MIN	Minimum selection
36	SEL	Binary selection

25.4 S5-S7 Converting Blocks

Floating-point arithmetic

FC	Name	Designation
61	GP_FPGP	Convert fixed-point to floating-point
62	GP_GPFP	Convert floating-point to fixed-point
63	GP_ADD	Add floating-point numbers
64	GP_SUB	Subtract floating-point numbers
65	GP_MUL	Multiply floating-point numbers
66	GP_DIV	Divide floating-point numbers
67	GP_VGL	Compare floating-point numbers
68	GP_RAD	Find the square root of a floating-point number

Signal functions

FC	Name	Designation
69	MLD_TG	Clock pulse generator
70	MLD_TGZ	Clock pulse generator with timer function
71	MLD_EZW	Initial value single blinking wordwise
72	MLD_EDW	Initial value double blinking wordwise
73	MLD_SAMW	Group signal wordwise
74	MLD_SAM	Group signal
75	MLD_EZ	Initial value single blinking
76	MLD_ED	Initial value double blinking
77	MLD_EZWK	Initial value single blinking (wordwise) memory bit
78	MLD_EZDK	Initial value double blinking (wordwise) memory bit
79	MLD_EZK	Initial value single blinking memory bit
80	MLD_EDK	Initial value double blinking memory bit

Integrated functions

FC	Name	Designation
81	COD_B4	BCD-binary conversion 4 decades
82	COD_16	Binary-BCD conversion 4 decades
83	MUL_16	16-bit fixed-point multiplier
84	DIV_16	16-bit fixed-point divider

Basic functions

FC	Name	Designation
85	ADD_32	32-bit fixed-point adder
86	SUB_32	32-bit fixed-point subtractor
87	MUL_32	32-bit fixed-point multiplier
88	DIV_32	32-bit fixed-point divider

89	RAD_16	16-bit fixed-point square root extractor
90	REG_SCHB	Bitwise shift register
91	REG_SCHW	Wordwise shift register
92	REG_FIFO	Buffer (FIFO)
93	REG_LIFO	Stack (LIFO)
94	DB_COPY1	Copy data area (direct)
95	DB_COPY2	Copy data area (indirect)
96	RETTEN	Save scratchpad memory (S5-155U)
97	LADEN	Load scratchpad memory (S5-155U)
98	COD_B8	BCD-binary conversion 8 decades
99	COD_32	Binary-BCD conversion 8 decades

Analog functions

FC	Name	Designation
100	AE_460_1	Analog input module 460
101	AE_460_2	Analog input module 460
102	AE_463_1	Analog input module 463
103	AE_463_2	Analog input module 463
104	AE_464_1	Analog input module 464
105	AE_464_2	Analog input module 464
106	AE_466_1	Analog input module 466
107	AE_466_2	Analog input module 466
108	RLG_AA1	Analog output module
109	RLG_AA2	Analog output module
110	PER_ET1	ET100 distributed I/O
111	PER_ET2	ET100 distributed I/O

Math functions

FC	Name	Designation
112	SINUS	Sine
113	COSINUS	Cosine
114	TANGENS	Tangent
115	COTANG	Cotangent
116	ARCSIN	Arc sine
117	ARCCOS	Arc cosine
118	ARCTAN	Arc tangent
119	ARCCOT	Arc cotangent
120	LN_X	Natural logarithm
121	LG_X	Logarithm to base 10
122	B_LOG_X	Logarithm to any base
123	E_H_N	Exponential function with base e
124	ZEHN_H_N	Exponential function with base 10
125	A2_H_A1	Exponential function with any base

25.5 TI-S7 Converting Blocks

FB	Name	Designation
80	LEAD_LAG	Lead/lag algorithm
81	DCAT	Discrete control time interrupt
82	MCAT	Motor control time interrupt
83	IMC	Index matrix comparison
84	SMC	Matrix scanner
85	DRUM	Event maskable drum
86	PACK	Collect/distribute table data

FC	Name	Designation
80	TONR	Latching ON delay
81	IBLKMOV	Transfer data area indirectly
82	RSET	Reset process image bit by bit
83	SET	Set process image bit by bit
84	ATT	Enter value in table
85	FIFO	Output first value in table
86	TBL_FIND	Find value in table
87	LIFO	Output last value in table
88	TBL	Execute table operation
89	TBL_WRD	Copy value from the table
90	WSR	Save datum
91	WRD_TBL	Combine table element
92	SHRB	Shift bit in bit shift register
93	SEG	Bit pattern for 7-segment display
94	ATH	ASCII-hexadecimal conversion
95	HTA	Hexadecimal-ASCII conversion
96	ENCO	Least significant set bit
97	DECO	Set bit in word
98	BCDCPL	Generate ten's complement
99	BITSUM	Count set bits
100	RSETI	Reset PQ byte by byte
101	SETI	Set PQ byte by byte

102	DEV	Calculate standard deviation
103	CDT	Correlated data tables
104	TBL_TBL	Table combination
105	SCALE	Scale values
106	UNSCALE	Unscale values

25.6 PID Control Blocks

FB	Name	Designation
41	CONT_C	Continuous control
42	CONT_S	Step control
43	PULSGEN	Generate pulse
58	TCONT_CP	Continuous temperature control
59	TCONT_S	Step temperature control

25.7 Communication Blocks

FB	Name	Designation
8	USEND	Uncoordinated send
9	URCV	Uncoordinated receive
12	BSEND	Block-oriented send
13	BRCV	Block-oriented receive
14	GET	Read data from partner
15	PUT	Write data to partner

FC	Name	Designation
1	DP_SEND	Send data
2	DP_RECV	Receive data
3	DP_DIAG	Diagnostics
4	DP_CTRL	Control
62	C_CNTR	Scan connection status

For DP standard slaves and PROFINET IO devices

FB	Name	Designation
20	GETIO	Read inputs
21	SETIO	Set outputs
22	GETIO_PA	Read inputs consistently
23	SETIO_PA	Set outputs consistently

IE communications

FB	Name	Designation
63	TSEND	Send data
64	TRCV	Receive data
65	TCON	Establish connection
66	TDISCON	Cancel connection
67	TUSEND	Send data via UDP
68	TURCV	Receive data via UDP

UDT	Name	Designation
65	TCON_PAR	Data structure for connection configuration
651	TCON_PAR	TCP_conn_active
652	TCON_PAR	TCP_conn_passive
653	TCON_PAR	ISOonTCP_conn_active
654	TCON_PAR	ISOonTCP_conn_passive
655	TCON_PAR	ISOonTCP_conn_CP_active
656	TCON_PAT	ISOonTCP_conn_CP_passive
657	TCPN_PAR	UDP_local_open
66	TADD_PAR	Address structure of the communication partner
661	TADD_PAR	UDP_rem_address and port

25.8 Miscellaneous Blocks

FC	Name	Designation
60	LOC_TIME	Read local time and summer ID
61	BT_LT	Convert module time into local time
62	LT_BT	Convert local time into module time
63	S_LTINT	Set time interrupt according to local time

FB	Name	Designation
60	SET_SW	Switch over summer/winter time
61	SET_SW_S	Switch over summer/winter time with time status
62	TIMESTMP	Send messages with time stamp

UDT	Name	Designation
60	WS_RULES	Rules for summer/winter time switching

25.9 SIMATIC_NET_CP

Library program *CP 300*

FB	Name	Designation
		FMS communications:
2	IDENT	Identify partner
3	READ	Read data from partner
4	REPORT	Transmit variables
5	STATUS	Request status information from partner
6	WRITE	Write data to partner
8	USEND	Uncoordinated send
9	URCV	Uncoordinated receive
12	BSEND	Block-oriented send
13	BRCV	Block-oriented receive
14	GET	Read data from partner
15	PUT	Write data to partner
55	IP_CONF	Send configuration

FC	Name	Designation
1	DP_SEND	Send data
2	DP_RECV	Receive data
3	DP_DIAG	Diagnostics
4	DP_CTRL	Control
5	AG_SEND	Send data (PROFIBUS FDL and Industrial Ethernet)
6	AG_RECV	Receive data (PROFIBUS FDL and Industrial Ethernet)
7	AG_LOCK	Disable data exchange (Industrial Ethernet)
8	AG_UNLOC	Enable data exchange (Industrial Ethernet)
40	FTP_CONN	Establish connection to server
41	FTP_STOR	Send data block to server
42	FTP_RETR	Send file to client
43	FTP_DELE	Delete file on server
44	FTP_QUIT	Cancel connection
62	C_CNTRL	Scan connection status

UDT	Name	Designation
1	-	FILE_DB_HEADER

Library program *CP 400*

FB	Name	Designation
		FMS communications:
2	IDENT	Identify partner
3	READ	Read data from partner
4	REPORT	Send variables
5	STATUS	Request status information from partner
6	WRITE	Write data to partner
55	IP_CONF	Send connection configuration

FC	Name	Designation
5	AG_SEND	Send data (PROFIBUS FDL and Industrial Ethernet)
6	AG_RECV	Receive data (PROFIBUS FDL and Industrial Ethernet)
7	AG_LOCK	Disable data exchange (Industrial Ethernet)

8	AG_UNLOC	Enable data exchange (Industrial Ethernet)
40	FTP_CONN	Establish connection to server
41	FTP_STOR	Transmit data block to server
42	FTP_RETR	Transmit file to client
43	FTP_DELE	Delete file on server
44	FTP_QUIT	Cancel connection
50	AG_LSEND	Send data (TCP connections)
60	AD_LRECV	Receive data (TCP connections)

UDT	Name	Designation
1	-	FILE_DB_HEADER

25.10 Redundant IO (V1)

Support of redundancy for modules

Library program *Red_IO*

FB	Name	Designation
450	RED_IN	Read redundant I/O signals
451	RED_OUT	Output redundant I/O signals
452	RED_DIAG	Diagnose redundant I/O
452	RED_STAT	Read status of redundant I/O

FC	Name	Designation
450	RED_INIT	Initialize I/O redundancy
451	RED_DEPA	Trigger depassivation

25.11 Redundant IO CGP

Support of redundancy for individual module channels

Library program *Red_IO*

FB	Name	Designation
450	RED_IN	Read redundant I/O signals
451	RED_OUT	Output redundant I/O signals
452	RED_DIAG	Diagnose redundant I/O
452	RED_STAT	Read status of redundant I/O

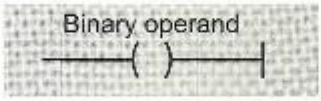
FC	Name	Designation
450	RED_INIT	Initialize I/O redundancy
451	RED_DEPA	Trigger depassivation

26 Function Set LAD

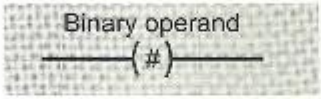
26.1 Basic Functions

Memory functions

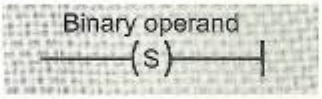
Singe coil



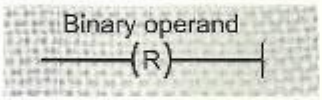
Midline output



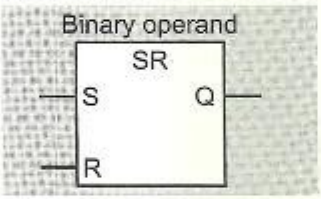
Set coil



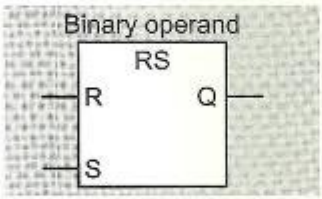
Reset coil



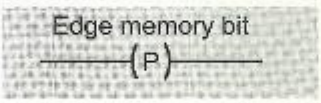
SR box



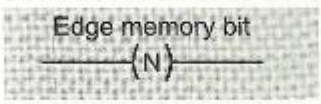
RS box



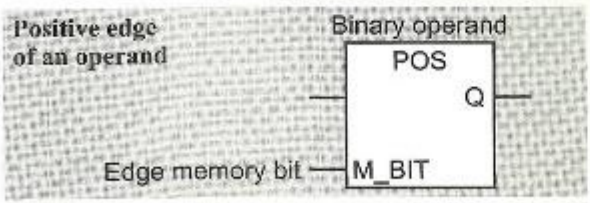
Positive edge in the power flow



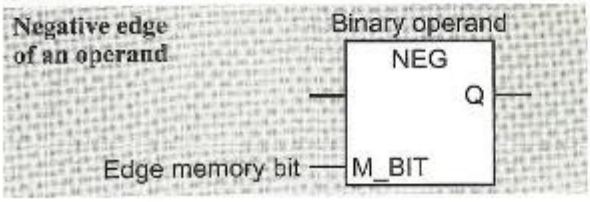
Negative edge in the power flow



Positive edge of an operand

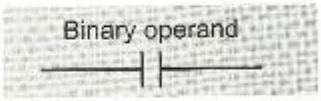


Negative edge of an operand



Binary checks and combinations

NO contact

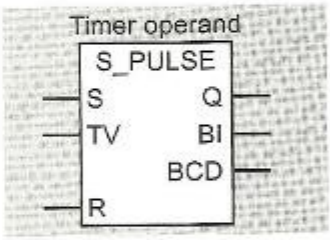


NC contact

NOT contact

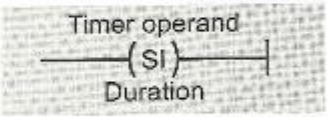
Timer functions

Timer box

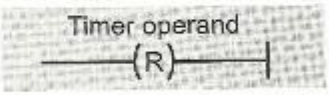


Individual elements

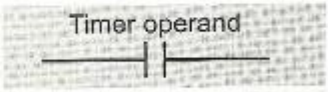
Start coil with time mode



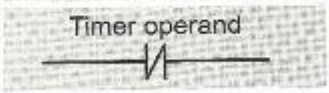
Reset coil



NO contact



NC contact

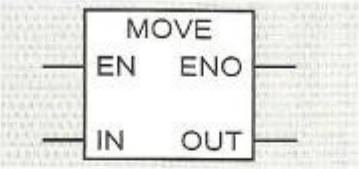


With the timer characteristics:

S_PULSE	SP	Pulse
S_PEXT	SE	Extended Pulse
S_ODT	SD	ON delay
S_ODTS	SS	Stored ON delay
S_OFFDT	SF	OFF delay

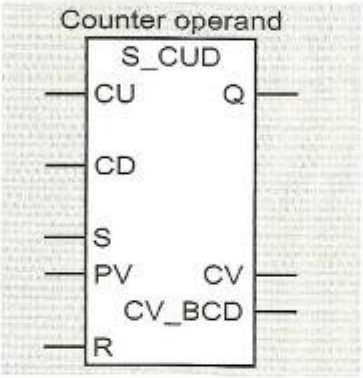
Transfer functions

MOVE box



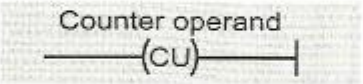
Counter functions

Counter box

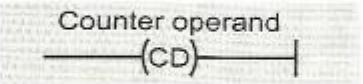


Individual elements

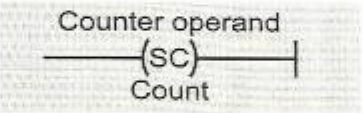
Count up coil



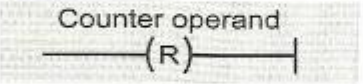
Count down coil



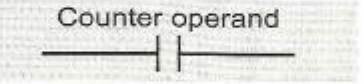
Set counter coil with count



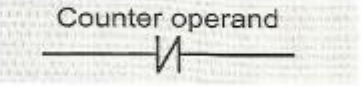
Reset coil



NO contact



NC contact



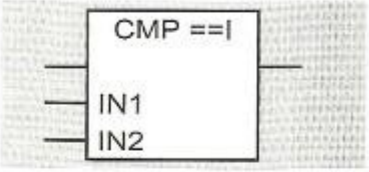
With the counter characteristics:

S_CUD	Up/down counter
S_CU	Up counter
S_CD	Down counter

26.2 Digital Functions

Comparison functions

Comparison box



Compare for	according to		
	INT	DINT	REAL
equal to	==I	==D	==R
not equal to	<>I	<>D	<>R
greater than	>I	>D	>R
greater than or equal to	>=I	>=D	>=R
less than	<I	<D	<R
less than or equal to	<=I	<=D	<=R

Arithmetic functions

Arithmetic box

Calculation	according to		
	INT	DINT	REAL
Addition	ADD_I	ADD_DI	ADD_R
Subtraction	SUB_I	SUB_DI	SUB_R
Multiplication	MUL_I	MUL_DI	MUL_R
Division	DIV_I	DIV_DI	DIV_R
Modulo	-	MOD_DI	-

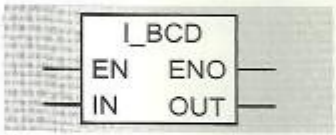
Mathematical Functions

Math box

SIN	Sine
COS	Cosine
TAN	Tangent
ASIN	Arc sine
ACOS	Arc cosine
ATAN	Arc tangent
SQR	Finding the square
SQRT	Finding the square root
EXP	Establishing the exponent
LN	Finding the logarithm

Conversion Functions

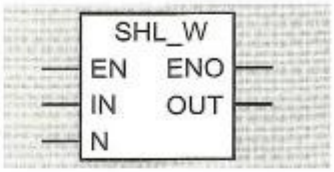
Conversion box



I_DI	Conversion of INT to DINT
I_BCD	Conversion of INT to BCD
DI_BCD	Conversion of DINT to BCD
DI_R	Conversion of DINT to REAL
BCD_I	Conversion of BCD to INT
BCD_DI	Conversion of BCD to DINT
	Conversion of REAL to DINT with rounding
CEIL	to next higher number
FLOOR	to next lower number
ROUND	to next whole number
TRUNC	without rounding
INV_I	INT one's complement
INV_DI	DINT one's complement
NEG_I	INT negation
NEG_DI	DINT negation
NEG_R	REAL negation
ABS	REAL absolute-value generation

Shift Functions

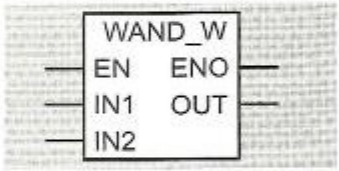
Shift box



SHL_W	Shift word left
SHL_DW	Shift doubleword left
SHR_W	Shift word right
SHR_DW	Shift doubleword right
SHR_I	Shift word with sign
SHR_DI	Shift doubleword with sign
ROL_DW	Rotate left
ROR_DW	Rotate right

Word Logic

Word logic box



WAND_W	AND word
WOR_W	OR word
WXOR_W	Exclusive OR word
WAND_DW	AND doubleword
WOR_DW	OR doubleword
WXOR_DW	Exclusive OR doubleword

26.3 Program Flow Control

Status Bits

Result greater than zero — >0

Result greater than or equal to zero — >=0

Result less than zero — <0

Result less than or equal to zero — <=0

Result not equal to zero — <>0

Result equal to zero — ==0

Result invalid (unordered) — UO

Overflow — OV

Stored overflow — OS

Binary result — BR

SAVE coil — (SAVE)

Jump Functions

Jump if RLO = "1" — Destination (JMP)

Jump if RLO = "0" — Destination (JMPN)

Entry, jump label — Destination

Master Control Relay

Activate MCR area — (MCRA)

Deactivate MCR area — (MCRD)

Open MCR zone — (MCR<)

Close MCR zone — (MCR>)

Block Functions

Calling a function block with data block — DB x, FB x (EN, ENO, IN1, OUT1, IN2, OUT2)

Calling a system function block with data block — DB x, SFB x (EN, ENO, IN1, OUT1, IN2, OUT2)

Calling a function block or a system function block as local instance — #name (EN, ENO, IN1, OUT1, IN2, OUT2)

Calling a function — FC x (EN, ENO, IN1, OUT1, IN2, OUT2)

Calling a system function — SFC x (EN, ENO, IN1, OUT1, IN2, OUT2)

Calling a non-parameterized function — FC x (CALL)

Calling a non-parameterized system function — SFC x (CALL)

RET coil, conditional block end — (RET)

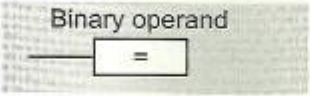
Open data block — DB x (OPN)

27 Function Set FBD

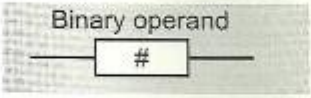
27.1 Basic Functions

Memory Functions

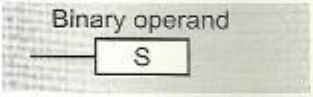
Assign



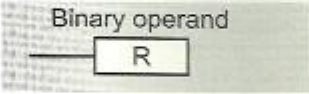
Midline output



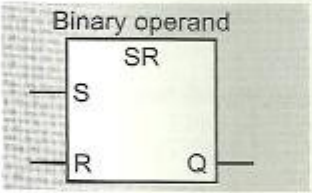
Set



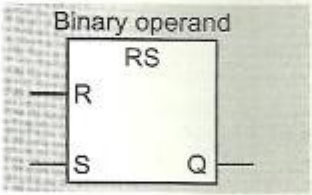
Reset



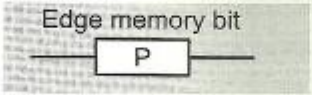
SR box



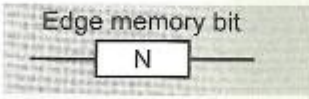
RS box



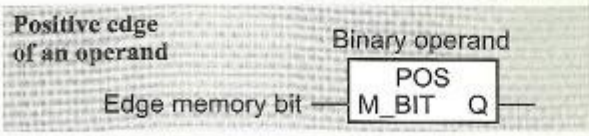
Positive edge of the RLO



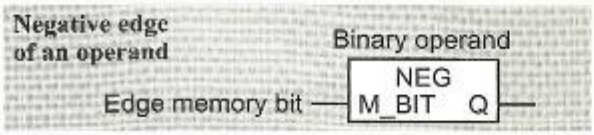
Negative edge of the RLO



Positive edge of an operand

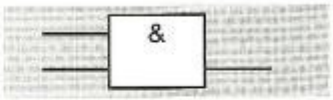


Negative edge of an operand

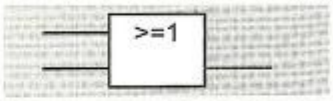


Binary Checks and Combinations

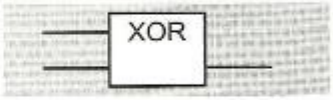
AND function



OR function



Exclusive-OR function

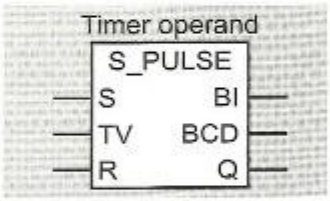


Check for signal state "1"

Check for signal state "0", negation

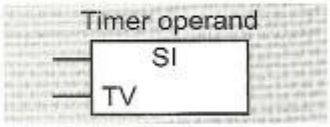
Timer Functions

Timer box

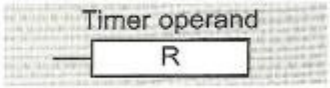


Individual elements

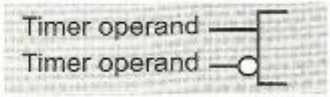
Start box with time mode



Reset box



Check timer status



With the timer characteristics:

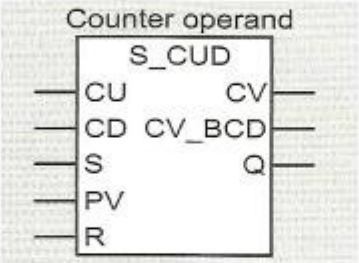
S_PULSE	SP	Pulse
S_PEXT	SE	Extended Pulse
S_ODT	SD	ON delay
S_ODTS	SS	Stored ON delay
S_OFFDT	SF	OFF delay

Transfer Functions

MOVE box

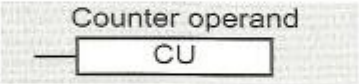
Counter Functions

Counter box

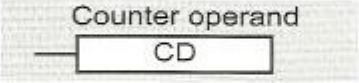


Individual elements

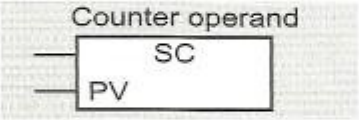
Count up box



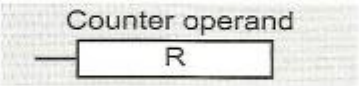
Count down box



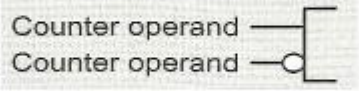
Set counter box with count



Reset box



Check counter status



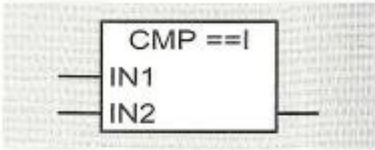
With the counter characteristics:

S_CUD	Up/down counter
S_CU	Up counter
S_CD	Down counter

27.2 Digital Functions

Comparison Functions

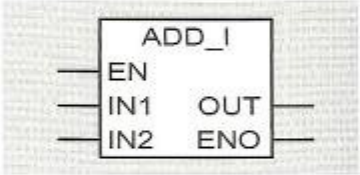
Comparison box



Compare for	according to		
	INT	DINT	REAL
equal to	==I	==D	==R
not equal to	<>I	<>D	<>R
greater than	>I	>D	>R
greater than or equal to	>=I	>=D	>=R
less than	<I	<D	<R
less than or equal to	<=I	<=D	<=R

Arithmetic functions

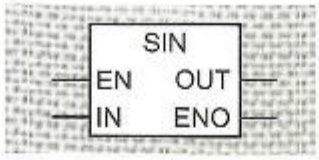
Arithmetic box



Calculation	according to		
	INT	DINT	REAL
Addition	ADD_I	ADD_DI	ADD_R
Subtraction	SUB_I	SUB_DI	SUB_R
Multiplication	MUL_I	MUL_DI	MUL_R
Division	DIV_I	DIV_DI	DIV_R
Modulo	-	MOD_DI	-

Mathematical Functions

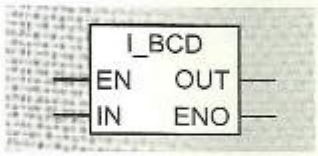
Math box



SIN	Sine
COS	Cosine
TAN	Tangent
ASIN	Arc sine
ACOS	Arc cosine
ATAN	Arc tangent
SQR	Finding the square
SQRT	Finding the square root
EXP	Establishing the exponent
LN	Finding the logarithm

Conversion Functions

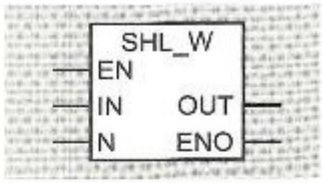
Conversion box



I_DI	Conversion of INT to DINT
I_BCD	Conversion of INT to BCD
DI_BCD	Conversion of DINT to BCD
DI_R	Conversion of DINT to REAL
BCD_I	Conversion of BCD to INT
BCD_DI	Conversion of BCD to DINT
	Conversion of REAL to DINT with rounding
CEIL	to next higher number
FLOOR	to next lower number
ROUND	to next whole number
TRUNC	without rounding
INV_I	INT one's complement
INV_DI	DINT one's complement
NEG_I	INT negation
NEG_DI	DINT negation
NEG_R	REAL negation
ABS	REAL absolute-value generation

Shift Functions

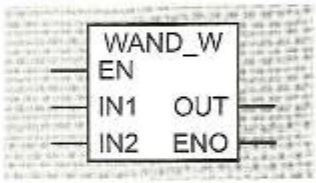
Shift box



SHL_W	Shift word left
SHL_DW	Shift doubleword left
SHR_W	Shift word right
SHR_DW	Shift doubleword right
SHR_I	Shift word with sign
SHR_DI	Shift doubleword with sign
ROL_DW	Rotate left
ROR_DW	Rotate right

Word Logic

Word logic box



WAND_W	AND word
WOR_W	OR word
WXOR_W	Exclusive OR word
WAND_DW	AND doubleword
WOR_DW	OR doubleword
WXOR_DW	Exclusive OR doubleword

27.3 Program Flow Control

Status Bits

Result greater than zero	>0
Result greater than or equal to zero	>=0
Result less than zero	<0
Result less than or equal to zero	<=0
Result not equal to zero	<>0
Result equal to zero	==0
Result invalid (unordered)	UO
Overflow	OV
Stored overflow	OS
Check binary result	BR
Save binary result	SAVE

Jump Functions

Jump if RLO = "1"	Destination JMP
Jump if RLO = "0"	Destination JMPN
Entry, jump label	Destination

Master Control Relay

Activate MCR area	MCRA
Open MCR zone	MCR<
Close MCR zone	MCR>
Deactivate MCR area	MCRD

Block Functions

Calling a function block with data block	DBx FBx EN OUT1 IN1 OUT2 IN2 ENO
Calling a system function block with data block	DBx SFBx EN OUT1 IN1 OUT2 IN2 ENO
Calling a function block or a system function block as local instance	#name EN OUT1 IN1 OUT2 IN2 ENO
Calling a function	FCx EN OUT1 IN1 OUT2 IN2 ENO
Calling a system function	SFCx EN OUT1 IN1 OUT2 IN2 ENO
Calling a non-parameterized function	FCx CALL
Calling a non-parameterized system function	SFCx CALL
Conditional block end	RET
Open data block	DB x OPN

Index

A

Actual parameters 257
Address priority 77
Addressing
  absolute 117
  indirect 408
  symbolic 118
AND function 136
ANY pointer
  description 410
  introduction 165
Arc functions 204
Archiving projects 56
Arithmetic functions 197
  status bits 222
ARRAY (data type) 126
Ascertaining a module address 383
Assign
  box (FBD) 146
Asynchronous errors 398
Automation License Manager 50

B

Background scanning OB 90 275
Binary flags 221
Binary logic operations 131
  FBD 134
Binary result 222
  EN/ENO 225
  SAVE 225
  status bit BR 222
Bit memory 48
Block functions for code blocks 235
Block libraries 415
Block parameters 252
  forwarding 260
Blocks
  checking block consistency 105
  comparing 76
  correcting interface 104
  description 98
  opening 106
  programming description 106
  programming FBD 112
  programming general 73
  programming LAD 110
  properties 101
  protection 103, 407
  structure 100
  transferring 83
  types 99
Box
  assign box 146
  memory box FBD 148
  memory box LAD 144
  reset box 148
  set box 148

C

CHAR (data type) 122
Checksum 58, 98
Clock memory 48
Coil
  reset coil 142
  set coil 142
  single coil 142
Cold restart 380
Combining connections 70
Comments multilingual 80
Communication
  distributed I/O 287
  global data communication 320
  IE communication 339
  introduction 35
  PtP communication 346
  S7 basic communication 324
  S7 communication 330
Communication error OB 87 399
Comparison functions 193
Compressing
  SFC 25 COMPRESS 282
  user program 84
Configuration in RUN 352
Configuration table 59
Configuring stations 58
Connecting a PLC 81

Connection table 65
Connections
configuring 65
loading data 68
Controlling I/O bits 233
Conversion functions 207
status bits 222
Counter functions
IEC counters 186
SIMATIC counters 182
CPU hardware faults OB 84 398
CPU information 82
Creating projects 54

D

Data block
functions 244
monitoring and modifying data addresses 92
offline/online 84
open 246
programming 113
registers 244
windows and views 114
Data operands 245
Data types
complex 125
elementary 120
subdivision 120
user-defined 128
DATE (data type) 125
DATE_AND_TIME (data type) 125
Device name, device number 305
Diagnosing hardware 86
Diagnostic interrupt OB 82 401
Diagnostics address 44
PROFIBUS DP 289
PROFINET IO 305
Digital flags 221
DINT (data type) 122
DINT calculation 200
status bits 222
Direct data exchange *see* DP master system
Disable output modules 377
Distributed I/O
addressing PROFIBUS DP 287
addressing PROFINET IO 304
configuring PROFIBUS DP 292
configuring PROFINET IO 306
introduction PROFIBUS DP 28
introduction PROFINET IO 31
system blocks 314
Down counting 185
DP master system
configuring 292
description 29
direct data exchange 301
equidistant bus cycles 301
in network configuration 64
isochronous mode 301, 302
DPV1 interrupts 367

E

Edge evaluation
description 152
Editor
FBD elements 112
LAD elements 110
program 73
symbolic 71
EN/ENO mechanism 225
Enabling peripheral outputs 90
Error handling 393
Example
binary scaler 154
conveyor belt 260
conveyor control 156
counter control 188
editing the message frame 413
feed 262
indirect copying of a data area 413
message frame 411
parts counter 261
save message frame 414
time-of-day check 411
Exclusive OR function 138
Exponentiation 205
Extended pulse timer 176

F

Fault-tolerant SIMATIC 24
First check 221
Forcing variables 88
Formal parameters 255
Function set
FBD 429
LAD 425

G

Gateways 68
Global data communication 320
Global data table 322

GSD files
  for PROFIBUS DP 299
  for PROFINET IO 309

H

Hardware catalog 59
Hardware configuration 58
Hardware interrupts 365

I

IEC counter functions 186
IEC functions
  library 419
IEC timer functions 180
Indirect addressing 408
Inputs 46
Insert/remove module interrupt OB 83 398
INT (data type) 122
INT calculation 199
  status bits 222
Interrupt handling 356
  DPV1 interrupts 367
  general remarks 356
  hardware interrupts 365
  multiprocessor interrupt 369
  synchronous cycle interrupts 370
  time-delay interrupts 360
  time-of-day interrupts 357
  watchdog interrupts 363
IP address 305
Isochronous mode 301

J

Jump functions 227

L

Language setting 79
Libraries
  communication blocks 422
  creating 56
  general 53
  IEC function blocks 419
  miscellaneous blocks 423
  organization blocks 415
  overview 415
  PID control blocks 422
  redundant IO (V1) 424
  redundant IO CGP 424
  S5-S7 converting blocks 420
  SIMATIC_NET_CP 423
  system function blocks 416
  TI-S7 converting blocks 421
Load memory 26
Local data
  static 241
  temporary 240
Local instances 242
Local time 278
Logarithm 205
Logical address 44

M

Main program OB1 270
Master Control Relay (MCR) 230
Mathematical functions 202
  status bits 222
Memory bits 48
Memory box
  FBD 148
  LAD 144
Memory card 25
Memory functions
  FBD 146
  LAD 142
Memory reset 378
Micro memory card 25
Midline outputs 150
Minimum scan cycle time 275
Modifying variables 88
Module time 278
Modules
  ascertaining addresses 383
  monitoring and modifying 61
  parameterizing 386
  start address 44
Monitoring variables 88
MOVE box 162
Move functions 161
Multilingual texts 80
Multiple instance *see Local instances*
Multiprocessor
  interrupt 369
  operation 282
Multiproject
  adjust projects 69
  creating and editing 57

N

NC contact 131
Negation
  conversion functions 212

normally closed (NC) contact 131
NOT contact (LAD) 134
RLO 139
scanning for "0" (FBD) 135
Network
configuration 62
templates 109
NO contact 131
Number range overflow 222

O

off-delay timer 179
on-delay timer 177
Online help 54
Open data block 246
Operating modes
HOLD 377
RESTART 380
RUN (main program) 270
START-UP 380
STOP 378
Operating modes of the CPU 376
OR function 137
Organization blocks
asynchronous errors OB 80 to OB 87 398
background scanning OB 90 275
determining runtime 283
diagnostic interrupt OB 82 401
interrupts OB 10 to OB 64 356
main program OB 1 270
overview 97
redundancy error OB 70 to 73 399
restart OB 100 to OB 102 377
synchronous errors OB 121, OB 122 393
Outputs 47
Overflow 222

P

Parallel circuits 132
Parameter types 128
Parameterizing modules 386
Peripheral inputs 45
Peripheral outputs 46
Pointer
general remarks 408
Power supply errors OB 81 398
Priority classes 95
Process image
description 46
isochrone updating 371
subprocess images 272
updating 272
Processing abort OB 88 399
PROFIBUS DP
addressing 287
configuring 292
isochronous mode 302
PROFINET IO
addressing 304
configuring 306
send cycle time/update time 310
sync domain 311
topology editor 311
PROFINET IO system
description 31
in network configuration 64
Program editor
FBD elements 112
general 73
LAD elements 110
Program elements catalog 108
Program execution errors OB 85 399
Program length 97
Program organization 271
Program processing methods 94
Program status 90
Program structure 270
Programming code blocks 106
Programming data blocks 244
Programming networks 108
Project
archiving 56
creating 54
general 53
object hierarchy 52
Project versions 57
Protecting
user program 286
PtP communication 346
Pulse timer 175

R

Rack failure OB 86 399
REAL (data type) 124
REAL calculation 200
status bits 222
Reference data 78
Reset box 148
Reset coil 142
Reset function 148
Response time 276
Restart characteristics 376

Restart types 380
Result of the logic operation 221
negation (FBD) 139
negation (LAD) 134
Retentive on-delay timer 178
Retentivity 379
Rewiring 76
Rotate functions 216
Rounding 210
RS memory function
FBD 150
LAD 144
Run-time meter 280

S

S5-S7 converting blocks
libraries 420
S5TIME (data type) 124
S7 basic communication
external 327
internal 324
S7 communication 330
S7-300 station 20
S7-400 station 22
Safety-related SIMATIC 24
Scan cycle monitoring time 274
Scan cycle statistics 274
Sensor type 139
Series circuits 132
Set box 148
Set coil 142
SFB 0 CTU 187
SFB 1 CTD 187
SFB 12 BSEND 333
SFB 13 BRCV 333
SFB 14 GET 334
SFB 15 PUT 334
SFB 16 PRINT 335
SFB 19 START 336
SFB 2 CTUD 187
SFB 20 STOP 336
SFB 21 RESUME 336
SFB 22 STATUS 337
SFB 23 USTATUS 337
SFB 3 TP 180
SFB 4 TON 180
SFB 5 TOF 180
SFB 52 RDREC 390
SFB 53 WRREC 391
SFB 54 RALRM 374
SFB 60 SEND_PTP 348
SFB 61 RCV_PTP 348
SFB 62 RES_RCVB 349
SFB 63 SEND_RK 350
SFB 64 FETCH_RK 351
SFB 65 SERVE_RK 351
SFB 75 SALRM 316
SFB 8 USEND 332
SFB 81 RD_DPAR 388
SFB 9 URC 332
SFC 0 SET_CLK 278
SFC 1 READ_CLK 278
SFC 100 SET_CLKS 278
SFC 101 RTM 281
SFC 102 RD_DPARA 388
SFC 103 DP_TOPOL 320
SFC 104 CIR 355
SFC 109 PROTECT 286
SFC 11 DPSYN_FR 318
SFC 12 D_ACT_DP 319
SFC 126 SYNC_PI 371
SFC 127 SYNC_PO 371
SFC 13 DPMRM_DG 319
SFC 14 DPRD_DAT 319
SFC 15 DPWR_DAT 320
SFC 2 SET_RTM 281
SFC 20 BLKMOV 165
SFC 21 FILL 165
SFC 22 CREAT_DB 248
SFC 23 DEL_DB 248
SFC 24 TEST_DB 248
SFC 25 COMPRESS 282
SFC 26 UPDAT_PI 273
SFC 27 UPDAT_PO 273
SFC 28 SET_TINT 359
SFC 29 CAN_TINT 360
SFC 3 CTRL_RTM 281
SFC 30 ACT_TINT 360
SFC 31 QRY_TINT 360
SFC 32 SRT_DINT 362
SFC 33 CAN_DINT 363
SFC 34 QRY_DINT 363
SFC 35 MP_ALM 370
SFC 36 MSK_FLT 397
SFC 37 DMSK_FLT 397
SFC 38 READ_ERR 397
SFC 39 DIS_IRT 372
SFC 4 READ_RTM 281
SFC 40 EN_IRT 373
SFC 41 DIS_AIRT 373
SFC 42 EN_AIRT 374
SFC 43 RE_TRIGR 274

SFC 44 REPL_VAL 397
SFC 46 STP 282
SFC 47 WAIT 282
SFC 48 SNC_RTCB 278
SFC 49 LGC_GADR 384
SFC 5 GADR_LGC 384
SFC 50 RD_LGADR 386
SFC 51 RDSYSST 403
SFC 52 WR_USMSG 400
SFC 54 RD_DPARM 389
SFC 55 WR_PARM 389
SFC 56 WR_DPARM 390
SFC 57 PARM_MOD 390
SFC 58 WR_REC 392
SFC 59 RD_REC 391
SFC 6 RD_SINFO 277
SFC 60 GD_SND 324
SFC 61 GD_RCV 324
SFC 62 CONTROL 337
SFC 64 TIME_TCK 280
SFC 65 X_SEND 328
SFC 66 X_RCV 329
SFC 67 X_GET 329
SFC 68 X_PUT 330
SFC 69 X_ABORT 330
SFC 7 DP_PRAL 318
SFC 70 LOG_GEO 384
SFC 71 GEO_LOG 384
SFC 72 I_GET 326
SFC 73 I_PUT 327
SFC 74 I_ABORT 327
SFC 78 OB_RT 283
SFC 79 SET 233
SFC 80 RSET 233
SFC 81 UBLKMOV 165
SFC 82 CREA_DBL 248
SFC 83 READ_DBL 165
SFC 84 WRIT_DBL 165
SFC 85 CREA_DB 248
SFC 87 C_DIAG 338
Shift functions 213
  status bits 224
SIMATIC counter functions 182
SIMATIC manager 50
SIMATIC timer functions 170
Single coil 142
Slot address 43
Source files
  updating or generating 76
Square-root extraction 204
Squaring 204
SR memory function
  FBD 149
  LAD 144
Start information
  DPV1 interrupts 368
  hardware interrupts 366
  interrupt handling 357
  main program OB 1 276
  multiprocessor interrupt 369
  restart 377
  synchronous cycle interrupts 371
  temporary local data 241
  time-delay interrupts 361
  time-of-day interrupts 358
  watchdog interrupts 363
Static local data 241
Status bits
  binary result BR 222
  description 221
  evaluating 224
  first check 221
  OR 222
  overflow OV 222
  RLO 221
  setting 222
  status 221
  stored overflow OS 222
Stored overflow 222
STRING (data type) 126
STRUCT (data type) 128
Subnets 37
Subprocess images 272
Symbol table 71
SYNC/FREEZE 299
Synchronous cycle interrupts 370
Synchronous errors 393
System blocks
  description 99
System clock 280
System diagnostics 400
System function blocks
  libraries 416
System memory 27

T

Temporary local data 240
TIME (data type) 125
Time of day 278
Time stamp conflict 104
TIME_OF_DAY (data type) 125
Time-based reaction

off-delay timer SFB 180
on-delay timer SFB 180
pulse timer SFB 180
Time-delay interrupts 360
Time-of-day interrupts 357
Timer characteristics
extended pulse timer 176
off-delay timer 179
on-delay timer 177
pulse timer 175
retentive on-delay timer 178
Timer functions
IEC timers 180
SIMATIC timers 170
Timing errors OB 80 398
Trigonometric functions 204

U

UDT (data type) 128
Up counting 185
User blocks 99
User data area 45
User data types 128
User program
compressing 84
load 83
protection 82
testing 90

V

Variable declaration table 106
Variable table 87

W

Warm restart 382, 383
Watchdog interrupts 363
Word logic
description 217
status bits 224
Work memory 27

Abbreviations

AI	Analog input
AO	Analog output
AS	Automation system
AS-I	Actuator-sensor interface
BR	Binary result
CFC	Continuous function chart
CP	Communications processor
CPU	Central processing unit
DB	Data block
DI	Digital input (module)
DO	Digital output (module)
DP	Distributed I/0
DR	Data record
EPROM	Erasable programmable read-only memory
FB	Function block
FBD	Function block diagram
FC	Function call
FEPROM	Flash erasable programmable read-only memory
FM	Function module
IM	Interface module
LAD	Ladder diagram
MC	Memory card
MCR	Master control relay
MMC	Micro memory card
MPI	Multipoint interface
OB	Organization block
OP	Operator panel
PG	Programming device
PLC	Programmable controller
PS	Power supply
RAM	Random access memory
RLO	Result of logic operation
SCL	Structured control language
SDB	System data block
SFB	System function block
SFC	System function call
SM	Signal module
SSL	System status list
STL	Statement list
UDT	User data type
VAT	Variable table