SIEMENS

SINUMERIK 840D
C-PLC Programming

Description of Functions 03.96 Edition

Manufacturer Documentation

SIEMENS

SINUMERIK 840D

C-PLC Programming

Description of Functions

Manufacturer Documentation

Valid for

Control	*Software Version*
SINUMERIK 840D	3.2

03.96 Edition

SINUMERIK® documentation

Printing history

Brief details of this edition and previous editions are listed below.

The status of each edition is shown by the code in the "Remarks" column.

Status code in the "Remarks" column:

A	New documentation.
B	Unrevised reprint with new Order No.
C	Revised edition with new status. If factual changes have been made on the page since the last edition, this is indicated by a new edition coding in the header on that page.

Edition	**Order No..**	**Remarks**
04.95	6FC5297-2AB60-0BP0	**A**
03.96	6FC5297-3AB60-0BP0	**C**

Siemens quality for software and training
to DIN ISO 9001, Reg. No. 2160-01

Order No. 6FC5297-3AB60
Printed in the Federal Republic of Germany

Siemens-Aktiengesellschaft.

Contents

Overview 1

High-level language programming

As an alternative to programming with STEP 7, it is possible to generate high-level language applications in the ANSI-C language for the integrated PLC of the SINUMERIK 840D machine tool control system.

The development environment described below, which is based on Borland C for DOS and the CS7DLIB library, allows this type of program to be developed and subjected to preliminary tests on the PC in off-line operation.

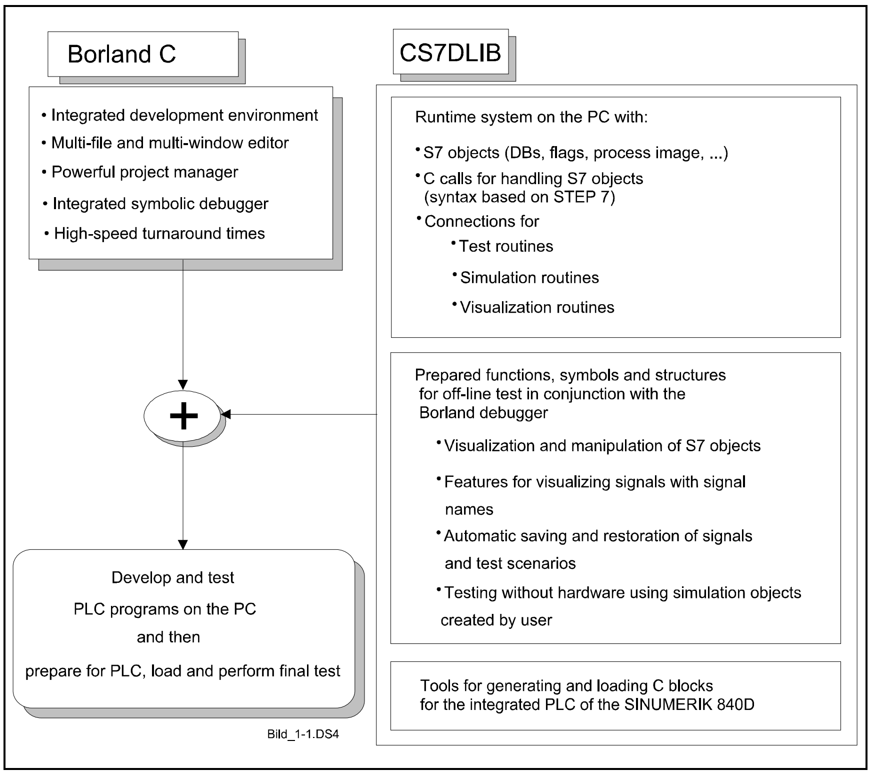

Borland C
• Integrated development environment
• Multi-file and multi-window editor
• Powerful project manager
• Integrated symbolic debugger
• High-speed turnaround times
+
Develop and test
PLC programs on the PC
and then
prepare for PLC, load and perform final test
CS7DLIB
Runtime system on the PC with:
• S7 objects (DBs, flags, process image, ...)
• C calls for handling S7 objects (syntax based on STEP 7)
• Connections for
• Test routines
• Simulation routines
• Visualization routines
Prepared functions, symbols and structures for off-line test in conjunction with the Borland debugger
• Visualization and manipulation of S7 objects
• Features for visualizing signals with signal names
• Automatic saving and restoration of signals and test scenarios
• Testing without hardware using simulation objects created by user
Tools for generating and loading C blocks for the integrated PLC of the SINUMERIK 840D
Bild_1-1.DS4

Fig. 1-1 Development of C-PLC programs with Borland C and CS7DLIB

Borland C package

The Borland C package provides you with powerful tools for generating, testing and modifying C programs.

CS7DLIB

The CS7DLIB library is used in conjunction with the package and provides functions which are relevant to runtime processes. These include timers, counters, I/O accessing functions as well as S7-specific data objects and system services.

CS7DLIB also offers a complete test sequence system which can manage user-specific routines for process simulation in addition to the actual PLC application. The status of the system objects can be saved and reconstructed.

In addition to the visualization facilities available with the Borland debugger, CS7DLIB provides user-defined screen pages for visualization of S7 objects in off-line test mode with which data contents with symbolic information can be displayed or manipulated.

Creating C blocks

The BSO tasking tool chain can be used to create C blocks from the program modules developed in off-line mode. These blocks can be loaded to the integrated PLC via the MPI interface.

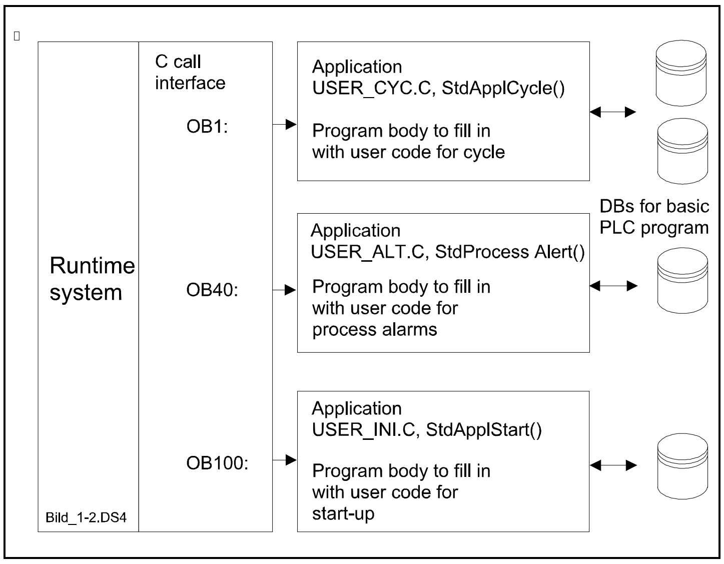

Runtime system
C call interface
OB1:
OB40:
OB100:
Application USER_CYC.C, StdApplCycle()
Program body to fill in with user code for cycle
Application USER_ALT.C, StdProcess Alert()
Program body to fill in with user code for process alarms
Application USER_INI.C, StdApplStart()
Program body to fill in with user code for start-up
DBs for basic PLC program
Bild_1-2.DS4

Fig. 1-2 Call interface for the C block

Basic PLC program

The basis of the PLC environment is the supplied basic STL program which links the PLC with the runtime system and the process (NCK-PLC interface) (see Fig. 1-2 Call interface for the C block).

Call interfaces (SFC63) for C blocks are available on the STEP 7 program execution levels "Basic cycle" (OB1), "Delay interrupt" (OB20), "Watchdog alarm" (OB35), "Process interrupt" (OB40) and "Start-up branch" (OB100). It is therefore possible to implement all user-specific expansions fully in C.

Data exchange between STEP 7 and C

Data are exchanged between STEP 7 and C by means of data blocks or bit memories. A data exchange may be required, for example, for operator communication and monitoring or for the NCK-PLC interface.

On-line test environment

Testing and start-up of the C program block in the control are supported by a C source level monitor for the PLC environment which is connected to the MPI interface.

Components and Installation 2

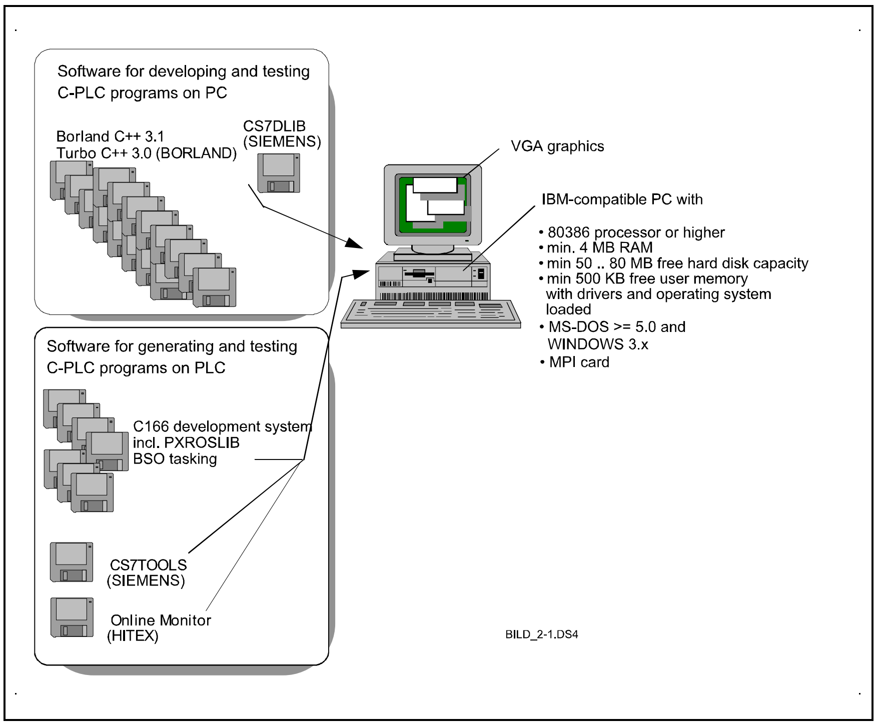

Software for developing and testing C-PLC programs on PC
Borland C++ 3.1
Turbo C++ 3.0 (BORLAND)
CS7DLIB (SIEMENS)
Software for generating and testing C-PLC programs on PLC
C166 development system incl. PXROSLIB BSO tasking
CS7TOOLS (SIEMENS)
Online Monitor (HITEX)
VGA graphics
IBM-compatible PC with
• 80386 processor or higher
• min. 4 MB RAM
• min 50 .. 80 MB free hard disk capacity
• min 500 KB free user memory with drivers and operating system loaded
• MS-DOS >= 5.0 and WINDOWS 3.x
• MPI card
BILD_2-1.DS4

Fig. 2-1 Components for the development of C-PLC programs

Software for developing and testing C-PLC programs on PC

See Section to 2.1 Development environment for PC

Software for generating and testing C-PLC programs on PLC

See Section to 2.2 Development environment for PLC

2.1 Development environment for PC

You will require the following items to develop and test C-PLC programs on the PC:

- C development package, **Borland C 3.0** or **3.1** (with DOS component) or **Turbo C 3.0** for off-line environment
- **CS7DLIB** extension from SIEMENS

C development package, Borland C 3.x / Turbo C 3.0

In order to install the development system, you must follow the installation instructions in the documentation of the development system. You do not need to install the Windows components, the class libraries or their on-line documentation, as only the DOS components are required. Nor do you need to install the source text of the libraries supplied by Borland. To avoid having to make changes to the project file supplied with the CS7DLIB software package, you should install the development system under directory C:\BORLANDC. Otherwise, you must change the path name for the include and library directories to your path in menu Options | Directories.

Installing CS7DLIB

After inserting the installation diskette into drive A:, install CS7DLIB by entering the following command

Enter: a:\>install *TargetDirectory*

Please enter a directory name of your choice for the *TargetDirectory* parameter, and specify the destination drive.

Example: a:\>install C:\CS7DLIB

This command copies the supplied files from the installation diskette plus all subdirectories into the specified directory.

Generating example project

The supplied example project can be generated from the directory created in the above operation.

Enter: c:\cs7dlib\>instdemo *DemoDirectory*

You must specify a directory name for the *DemoDirectory* parameter.

Example: *c:\cs7dlib*\>instdemo *cs7_demo*

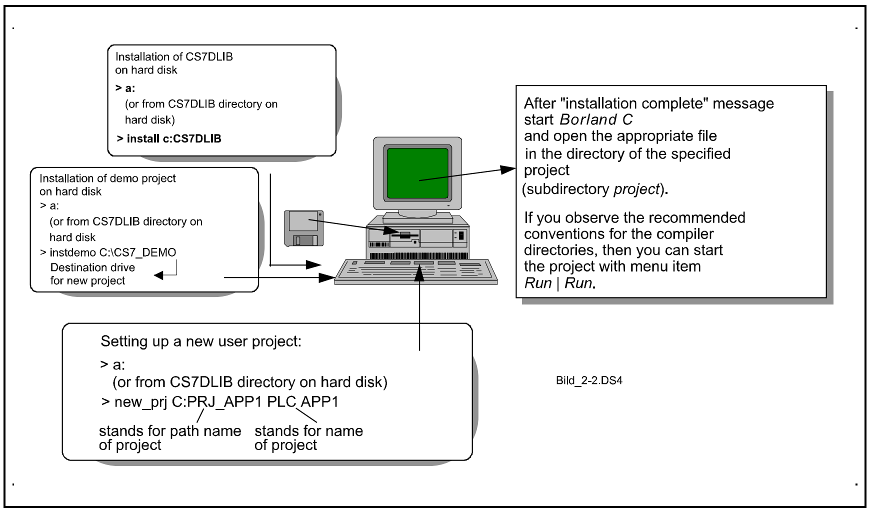

Installation of CS7DLIB
on hard disk
> a:
(or from CS7DLIB directory on
hard disk)
> install c:CS7DLIB
Installation of demo project
on hard disk
> a:
(or from CS7DLIB directory on
hard disk
> instdemo C:\CS7_DEMO
Destination drive
for new project
After "installation complete" message
start Borland C
and open the appropriate file
in the directory of the specified
project
(subdirectory project).
If you observe the recommended
conventions for the compiler
directories, then you can start
the project with menu item
Run | Run.
Setting up a new user project:
> a:
(or from CS7DLIB directory on hard disk)
> new_prj C:PRJ_APP1 PLC APP1
stands for path name
of project
stands for name
of project
Bild_2-2.DS4

Fig. 2-2 Installation of CS7DLIB and example project - Setting up a user project

---

Note

This version is based on the assumption that Borland C has been installed in directory *C:\BORLANDC*. If this is not the case, then the entries in menu Options | Directories must be changed to the actual path of the Borland compiler or else header files will not be found during compilation and library files of the Borland package will not be found during linking.

Setting up a new project

You can set up a new project as follows:

Enter: c:\cs7dlib\>new_prj c:\ProjectDirectory ProjectName

The path name of the new project must be specified in the *ProjectDirectory* parameter, and the project name must be specified in *ProjectName*.

Example: c:\cs7dlib\>new_prj c:\wzm_plc std_plcp

The project file you have now created is assigned the project name you have specified; all subdirectories required for the project are set up and the necessary files stored in the directory with the name entered above.

Development environment settings

The Borland C++ development system offers numerous options which you can set via menus.

Please note the following points:

- The CS7DLIB requires the large memory model of Borland C++ and also expects a DOS-EXE file to be generated.
- The two paths ..\SYS_INC and ..\USR_INC are entered as include directories for the project, in addition to the Borland path.
- The .OBJ and .EXE files of the project are stored in the ..\BIN project directory.
- The ..\PROJECT project directory is used as the starting point for calling up the development environment.

2.2 Development environment for PLC

The following items are required for generating and loading a C block on the PLC:

C166 development system

BSO tasking tools, version 4.0, development package for microprocessor types SAB 80C165/80C166. This package contains C compilers, assemblers, linkers and the required libraries. The library expansion PXROSLIB (C166 Special Stack Frame Library) must also be installed. Please follow the installation instructions given in

References: /BSO/, Users Guide

On-line monitor

HITEX user interface (space requirement: approx 3 Mbytes, HITEX licence). This package also includes the symbol preprocessor SP166TA.EXE (see Section 3.4 On-line monitor). Please follow the installation instructions given in

References: /HITEX/, Users Manual

Directory \CS7TOOLS\HITEX must be specified as the destination directory for installation purposes. In addition, symbol preprocessor SP166TA.EXE must be copied to directory \CS7TOOLS.

Runtime library

CS7RTLIB.LNO, the runtime library for accessing S7 objects from C programs (disk 1,directory \CS7DLIB\LIB), incl. the call interfaces AB_START.OBJ and ABMAIN.OBJ.
CS7RTLIB.LNO, AB_START.OBJ, and ABMAIN.OBJ are stored in directory CS7DLIB\LIB (see Section 2.3 Overview of directory structure).

CS7TOOLS

Generation tools, loading tools and tools for on-line testing during start-up.

- MPI tools
  Directory \CS7TOOLS\MPI

MPIDOS.EXE	MPI driver
MPIMON.EXE	MPI driver
BT_L7STD.COM	MPI driver
BT_L7TSR.COM	MPI driver
NETNAMES.CPU	Default settings for the MPI interface
NC_CD.EXE	Directory change in the NCK
NC_DIR.EXE	Directory display in the NCK
COMON.BAT	Control file for installing the MPI drivers
COMOFF.BAT	Control file for de-installing the MPI drivers

- Generating tools
  Directory \CS7TOOL

AB_GEN.EXE	Generates a loadable C block
BS_ADDR.EXE	Locates code and data segments
RDDBBPLC.EXE	Reads the start address of the C block in the PLC

- Control files
  Directory \CS7TOOLS

CC.BAT	Control file for compiling the C source files

- Loading tools
  Directory \CS7TOOLS

DOWNPLC.EXE	Loads the C and STEP 7 blocks on the PLC
UPPLC.EXE	Saves the STEP7 blocks from the PLC

- Monitor tools:
  Directory \CS7TOOLS\HITEX

AB15.1V1	Monitor block
MONLOAD.BAT	Loads the monitor block on the PLC
HIT_167.CFG	Config file for HITEX user interface
DEBUGGER.INI	File with cross reference to Debug data block (DB71)
STARTUP.SCR	HiScript file which is executed on start of the HITEX user interface
MONSTART.BAT	Starts the HITEX user interface

- Basic PLC program for SINUMERIK 840D:
  Directory \CS7TOOLS\GP840D

AWLLOAD.BAT	Loads the basic PLC program on the PLC
AWLSAVE.BAT	Saves the basic PLC program from the PLC

Installing CS7TOOLS

To install CS7TOOLS, insert the installation diskette in drive A:

Enter: a:\>install *TargetDirectory*

Please enter a directory name of your choice for the *TargetDirectory* parameter, and specify the destination drive.

Example: a:\>install C:\CS7TOOLS

This command copies the supplied files from the installation diskette with all subdirectories to directory \CS7TOOLS in the specified drive.

Extending AUTOEXEC.BAT

The AUTOEXEC.BAT file must be extended as follows:

- Path name for BSO tools
  <LW>:\C166\BIN386
- Path name for CS7 tools
  <LW>:\CS7TOOLS
- Environment variables for include files
  set C166INC=<LW>:\C166\INCLUDE
- Environment variables for MPI working directory
  set TEMP=<LW>:\TEMP
- Environment variables in MPI driver directory
  set BTDIR=<LW>:*CS7TOOLS\MPI*
- Settings for DOS extender
  set DOS16M=11
  set DOS4GVM=@NEW4G.VMC
  set DOS4GPATH=<BSO main directory>\bin386\, e.g.
  c:\c166\bin386\

  References: /BSO/, Users Guide

2.3 Overview of directory structure

The following diagram shows you the directory structure of a C block project:

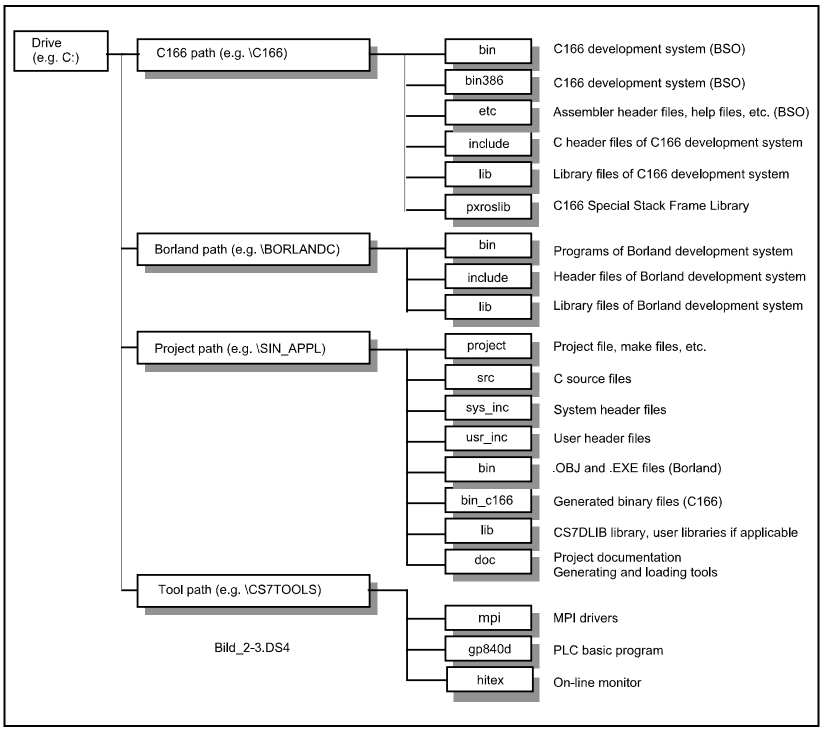

Drive (e.g. C:)
C166 path (e.g. \C166)
bin
C166 development system (BSO)
bin386
C166 development system (BSO)
etc
Assembler header files, help files, etc. (BSO)
include
C header files of C166 development system
lib
Library files of C166 development system
pxroslib
C166 Special Stack Frame Library
Borland path (e.g. \BORLANDC)
bin
Programs of Borland development system
include
Header files of Borland development system
lib
Library files of Borland development system
Project path (e.g. \SIN_APPL)
project
Project file, make files, etc.
src
C source files
sys_inc
System header files
usr_inc
User header files
bin
.OBJ and .EXE files (Borland)
bin_c166
Generated binary files (C166)
lib
CS7DLIB library, user libraries if applicable
doc
Project documentation
Generating and loading tools
Tool path (e.g. \CS7TOOLS)
mpi
MPI drivers
gp840d
PLC basic program
hitex
On-line monitor
Bild_2-3.DS4

Fig. 2-3 Overview of main directory paths

Description of individual directories

- C166 path, development system for the C166 processor which is required in order to generate the code executed on the PLC.
- Borland path, development system Borland C++ with which the C user programs are developed with the support of the CS7DLIB.
- Project path, directory for the current project. Each project should be set up as a separate directory tree (command *new_prj,* see Section 2.1 Development environment for PC).
- Tool path, tools for starting up and testing the C user programs on the PLC (MPI drivers, C block generating tools, loading tools, monitor tools, etc.).

2.4 Hardware

PC hardware requirements

- PC AT386 or higher with DOS >=5.0 and MPI card, VGA graphics
- Up to 80 Mbytes of free hard disk storage capacity
- Minimum of 4 Mbytes main memory
- Minimum of 500 Kbytes of free DOS memory

2.5 System resources of the PLC

The maximum available PLC memory is:

On the AS314:	672 Kbytes
On the AS315:	1280 Kbytes

C Block Programming 3

3.1 Conventions

3.1.1 Language and functionality

The full functionality in accordance with the ANSI-C standard is available for the C applications. Functions which require a specific type of environment (for example, output or file functions, etc.) and which cannot operate either in the S7 or in the PC test environment are naturally not permissible. Function expansions of the tasking C compiler specific to the 80165 cannot be tested with the off-line environment. It is not permissible to change the processor setting or to access the special function register of the SAB 80C165 microcontroller.

In addition to the ANSI-C vocabulary, a series of S7 basic instructions are also provided:

Process image functions

You can access the **process image** for read or write purposes either *bit by bit, byte by byte, word by word, or in doublewords*. You must distinguish between the input area of the process image (PII) and the output area of the process image (PIQ).

Table 3-1 Process image functions

Operation	**C function**

Read process input image (bit/byte/word/doubleword)	E_R()
	EB_R()
	EW_R()
	ED_R()

Write process input image (bit/byte/word/doubleword)	E_W()
	EB_W()
	EW_W()
	ED_W()

Determine address of PII (for pointer access)	ADR_PAE()

Read direct access to I/O inputs (bit/byte/word/doubleword)	L_PEB()
	L_PEW()
	L_PED()

Read process output image (bit/byte/word/doubleword)	A_R()
	AB_R
	AW_R()
	AD_R()

Write process output image (bit/byte/word/doubleword)	A_W()
	AB_W()
	AW_W()
	AD_W()

Determine address of PIQ (for pointer access)	ADR_PAA()

Write direct access to I/O outputs (byte/word/doubleword)	T_PAB()
	T_PAW()
	T_PAD()

Please note that a transfer takes place during the off-line test between the simulated I/O area and the process image depending on your input in the supplied source file user_cfg.c; in this case, you can determine the area for the output and the input yourself. For test purposes, it is often advisable to keep this area small in order to avoid undesirable side effects which may adversely affect the test.

Bit memory functions

SIMATIC S7 can make use of so-called bit memories, which are global variables, which can be written or read bit by bit, byte by byte, in 16-bit blocks or 32-bit blocks. The bit memories are addressed via the byte offset (parameter *s7_byte_offset*) referred to the 0 memory byte. In the case of bit-by-bit accessing, the bit offset (parameter *s7_bit_offset*) specifies the bit in the memory byte.

Table 3-2 Bit memory functions

Operation	C function

Read bit memory (bit/byte/word/doubleword)	M_R()
	MB_R()
	MW_R()
	MD_R()

Write bit memory
(bit/byte/word/doubleword)
	M_W()
	MB_W()
	MW_W()
	MD_W()

Determine address of bit memory area (for pointer access)	ADR_MRK()

Data block functions

Data blocks are global memory areas of a size specified by the user in each case. They are addressed by a data block number. Data blocks are required to exchange data via interfaces with external devices such as, for example, operator interfaces or the NCK.

In contrast to data blocks in the SIMATIC S7 environment which are loaded from a programming device/PC *or* generated in the program, the memory area for such blocks must *always* be allocated by the user in the development and test environment on the PC. The CS7DLIB automatically sets up the data blocks used as standard in the SINUMERIK 840D (see References /PLCGP).

References: /PLCGP/, Description of Functions: Standard Machine

Once a data block exists, it can be opened. The appropriate function call OPN_DB() supplies a "handle" in exchange which is required for all further calls of the data block. You can have several data blocks opened at the same time and access them optionally via the functions below if you are managing and using the appropriate handles.

Addressing within data blocks is implemented via the byte offset (parameter s7_byte_offset, referred to the 0 byte of the data block addressed by handle s7_db_handle). When blocks are accessed bit by bit, the bit offset (parameter s7_bit_offset) must also be specified.

Table 3-3 Data block functions

Operation	C function

Determine data block handle
Determine data block address (pointer access)
Determine data block length
	OPN_DB()
	ADR_DB()
	LNG_DB()

Read from DB
(bit/byte/word/DWORD)
	D_R()
	DB_R()
	DW_R()
	DD_R()

Write to DB
(bit/byte/word/DWORD)
	D_W()
	DB_W()
	DW_W()
	DD_W()

Performance can be improved by directly accessing the data blocks via pointers which can be used to address any block. No range or write protection checks are performed, however, so that the user must assure consistency (data structure, and in particular non-violation of range limits). The S7-specific byte order ("Big Endian") must be observed (see Fig. 3-1 SIMATIC byte order.). S7 data are always stored with the highest-order byte at the lowest address. Data block pointers are determined by means of the "ADR_DB" function.

Timer functions

SIMATIC S7 offers the programmer 5 different types of timer, each type with its own characteristics:

- The **pulse timer** is set to the specified value by a *positive edge* at logic input *rlo*. This value is counted down to 0 in the specified clock cycle. The logic output (return value of function) remains at 1 as long as the timer value is higher than 0. A *rlo* input value of 0 resets a pulse timer.
- The **timer with extended pulse** is set to the specified value by a *positive edge* at logic input *rlo*. This value to counted down to 0 in the specified clock cycle. The logic output (return value of function) remains at 1 as long as the timer value is higher than 0. A *rlo* input value of 0 does not reset the timer. The next positive edge at rlo resets the timer, which has still not expired, back to the specified value, i.e. it extends the pulse.
- The **timer with ON delay** is set to the specified value by a *positive edge* at logic input *rlo*. This value is counted down to 0 in the specified clock cycle. The logic output (return value of function) does not, however, switch to 1 until the time has expired and input rlo is still at 1. When rlo switches to 0, the output also switches to 0.
- The **timer with OFF delay** is set to the specified value by a *negative edge* at logic input *rlo*. This value is counted down to 0 in the specified clock cycle.
- The **timer with latched ON delay** is set to the specified value by a *positive edge* at logic input *rlo* This value is counted down to 0 in the specified clock cycle. The logic output (return value of function) does not, however, switch to 1 until the time has expired. It remains at 1 even when rlo switches to 0. The output state can be reset only by a reset command. For further details, please refer to

  References: /S7/, User Manual

Table 3-4 Timer functions

Operation (timer function)	**Function**
Timer	SP_T()
Extended pulse	SE_T()
ON delay	SD_T()
Latched ON delay, OFF delay	SS_T(), SA_T()
Timer reset	R_T()
Timer enable	F_T()
Scan timer value	LV_T()
Scan timer scale	LS_T()
Scan timer status	TS()

Counter functions

SIMATIC S7 can make use of counter objects which can be set on an edge-triggered basis and reset on a status-dependent basis. These objects permit edge-triggered up/down counting. It is possible to interrogate the counter contents (max. 999) and the counter status (counter contents > 0). For further details, please refer to

References: /S7/, User Manual

Special features regarding accessing of process images, bit memories and data blocks

Word-serial or doubleword-serial access operations by means of these access functions result in the C byte order (Little Endian Format) being converted to the SIMATIC Big Endian Format (see Fig. 3-1 SIMATIC byte order.). S7 data are always stored with the highest-order byte at the lowest address. The user should always use the defined access functions to transfer data to and from S7 objects.

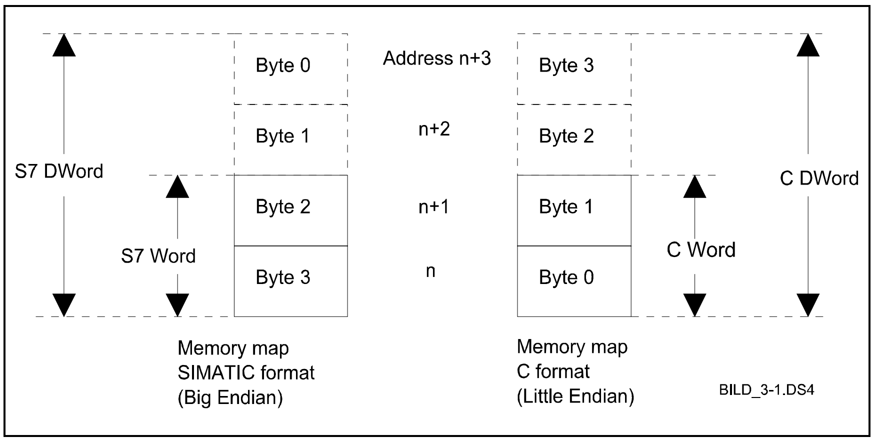


Fig. 3-1 SIMATIC byte order. S7 data are always stored with the highest-order byte at the lowest address

A bit-serial read access supplies the logic status of the appropriate bit. In the case of a write access, the addressed signal bit is set according to the logic state of the input value.

During processing (PC or PLC environment) the system detects whether the valid addressing space of an S7 object has been violated. This error is displayed in the status line in the PC environment; the PLC environment branches with an error identifier (and additional debug parameters) to an error handler which the user can program freely (see 5.1 Access to local data).

Note

Please refer to file *..\sys_inc\func_doc\clib_doc* for further details about transfer parameters and return values.

System services

A series of system services for manipulating the runtime system (enabling/disabling/initiating watchdog, process and delay alarm levels), for generating data blocks and for interrupting processing (STOP state) are also available.

For further details regarding transfer parameters and error messages, please refer to file *..\sys_inc\func_doc\clib_doc*.

Table 3-5 System services

Operation	C function
Generate data block	SFC_Create_DB()

Time-of-day alarm level
Parameterize/activate/deactivate
Status interrogation
(in preparation)
	SFC_Set_Time_Alarm()
	SFC_Activate_Time_Alarm()
	SFC_Cancel_Time_Alarm()
	SFC_Query_Time_Alarm()

Delay alarm level
Start/deactivate
Status interrogation
	SFC_Start_Del_Alarm()
	SFC_Cancel_Del_Alarm()
	SFC_Query_Del_Alarm()

Disable/enable alarm processing levels	SFC_Disable_Event_Processing()
	SFC_Enable_Event_Processing()

Delay/enable processing of existing alarms	SFC_Disable_Alarm_Interruption()
	SFC_Enable_Alarm_Interruption()

Retrigger cycle time (monitoring)	SFC_Retrigger()

Set system time
Read system time into clock structure
Interrogate system timer
(0 to 2**32-1 msec)
	SFC_Set_Clk()
	SFC_Read_Clk()
	SFC_Time_Tick()

Initiate STOP state	SFC_Stop()

Set operating hours counter
Start/stop operating hours counter
Read status of operating hours counter
	SFC_Set_Rtm()
	SFC_Ctrl_Rtm()
	SFC_Read_Rtm()

The functionality of the library functions is identical to that of the appropriate STEP 7 functions. Please refer to

References: /S7/, User Manual

Program execution levels

The following S7 program execution levels are available for the C program blocks. These levels are called up by the system according to defined events. Depending on the event or program execution level, one of the following functions is called by the system:

Table 3-6 Functions for program execution levels

Function name	Description	Cf. S7	Priority
StdApplCycle()	Free cycle	OB 1	1
StdTimeAlert()	Time-of-day alarm Function is called at user-defined time (in preparation)	OB 10	2
StdDelayedTimeAlert ()	Delay alarm Function is called after expiry of delay defined by user	OB 20	3
StdWatchdogAlert()	Watchdog alarm Function is called periodically according to user-defined time (default: 100 ms)	OB 35	12
StdProcessAlert()	Process alarm Call implemented by process signals (e.g. alarm module, M function transfer from NCK)	OB 40	16
StdApplStart()	Start-up Function is called on system start	OB 100	27

With regard to execution level priorities, the execution level with the highest ordinal number interrupts the level with the lowest ordinal number.

Return values

Please note that a number of the functions have a return type which starts with the prefix *F_* or *SFC_* . If you define a variable which receives the return value, then you should do this with the data type of the same name *without the prefix F_ or SFC_*. This convention must be observed in order to conceal the different subordinate addressing modes.

Please refer to the header file CLIB_DOC.H for further details on how to use these functions.

See Section to 3.2 Off-line program development
for details of functions for the off-line programming environment which are only available in the PC environment.

3.1.2 Use of data types

Data types

Within the scope of the CS7DLIB development package, a variety of precautions has been taken to ensure easy portability and to conceal system-specific differences between the PC and S7 environments. Important in this respect are the abstract data types used, some of which are derived from elementary data types and others user-compiled data types.

User data types

For your own applications, use only the data types provided by CS7DLIB or, if you create your own data types, make sure they are derived from these prespecified data types. This does not impose any restrictions on you in practical terms and also means that you do not need to bother with complicated, system-specific details. If you create a variable to use in calls of S7 utilities, then all you need to do is copy the appropriate parameters from the function prototypes of the CS7_CLIB.H header file and insert them in your source code.

Your source code will therefore be independent of the appropriate destination system and thus fully testable on the PC with CS7DLIB.

Storing data

Data are either declared in the C program modules or in S7 data blocks and bit memories. If data need to be available in the S7 world or via interfaces on external devices (e.g. for operator interfaces or NCK), then they must be stored in S7 data blocks or flags; the SIMATIC byte order (see Fig. 3-1 SIMATIC byte order.) must be observed in this case if applicable. S7 data are always stored with the highest-order byte at the lowest address.

Owing to the memory segmentation of the CPU 314 destination processor, the following must be noted with respect to global or static variables:

- Up to 32K so-called near data which can be accessed particularly quickly. Use the elementary data types UBYTE, WORD .. without prefix for these data (see Fig. 3-2 Overview of data types). Data objects within the near data area must not be larger than 16K.
- An "unlimited", so-called huge data area (extended memory area). Use of this area effects a slightly slower and more comprehensive code. Create this type of data using the elementary data types with the prefix G_, e.g. (G_UBYTE, G_WORD, see Fig. 3-2 Overview of data types).

If possible, access operations to huge data should be minimized by careful memory page allocation. The absolute memory address within these areas cannot be influenced by the user.

Caution

If data formats are modified or additional variables declared, the memory location of a data may be shifted uncontrollably.

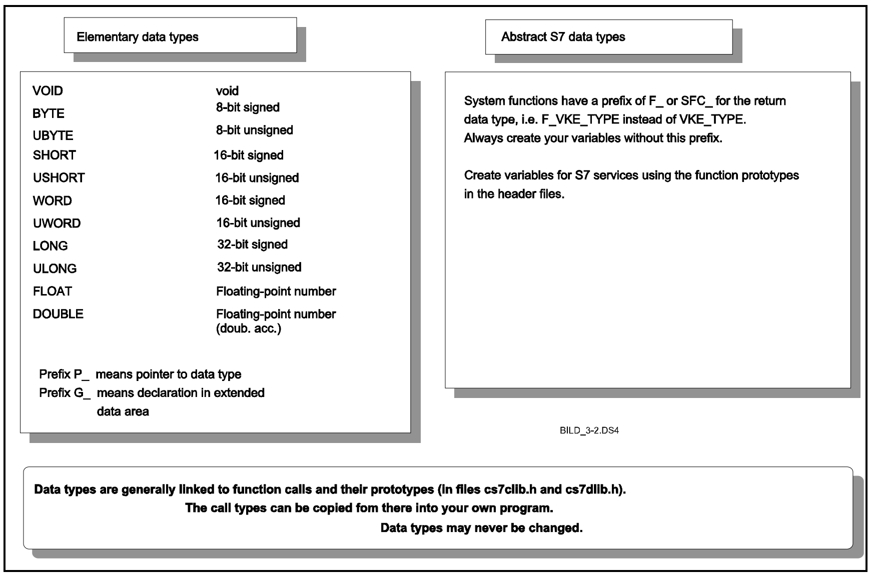
Reloading of C programs after changes to data declarations may therefore only take place in the PLC STOP state.

Initialize data

Static data are not stored on the stack, but in a defined memory area. This memory area is initialized during power-up:

- Data with a programmed start value are always initialized with this value during power-up.
- Data for which no initial value is specified are not initialized. These data therefore remain unchanged by a POWER-ON/RESET - caution when modifying the data declaration.

Observe the following list of data types which are relevant to you as the user (see also *..\sys_inc\data_def\datatype\datatype.h*):



Elementary data types

VOID	void
BYTE	8-bit signed
UBYTE	8-bit unsigned
SHORT	16-bit signed
USHORT	16-bit unsigned
WORD	16-bit signed
UWORD	16-bit unsigned
LONG	32-bit signed
ULONG	32-bit unsigned
FLOAT	Floating-point number
DOUBLE	Floating-point number (doub. acc.)

Prefix P_ means pointer to data type
Prefix G_ means declaration in extended data area

Abstract S7 data types

System functions have a prefix of F_ or SFC_ for the return data type, i.e. F_VKE_TYPE instead of VKE_TYPE.
Always create your variables without this prefix.

Create variables for S7 services using the function prototypes in the header files.

BILD_3-2.DS4

Data types are generally linked to function calls and their prototypes (in files cs7clib.h and cs7dlib.h).
The call types can be copied fom there into your own program.
Data types may never be changed.

Fig. 3-2 Overview of data types

3.1.3 Constants

Classification of constants

In order to parameterize functions or obtain function results, you require a variety of constants. These are divided into 2 groups:

- Constants for control or status signals, parameters and return values. Examples of these are the logic constants VKE_TRUE and VKE_FALSE, execution level IDs and STD_APPL_START_ID or transfer parameters to the system.
  (The constants are in *..\sys_inc\data_def\datatype\datatype.h).*
- Identifiers for visualizing S7 objects. Each S7 object has an identifier for bit display or for numerical display in some cases (see Section 3.2.4 User-defined visualization objects).
  (The constants are in *..\sys_inc\data_def\sys_objd.h*).

3.1.4 Runtime environment and standard program structure

A C application comprises the basic cycle, a range of event-driven or time-based runtime levels as well as an initialization phase.

The initialization phase is executed once before cyclic operation commences; processing of the basic cycle is then initiated or, if applicable, processing of the event-driven or time-based levels.

In addition, applications for the off-line test environment can be extended (see Section 3.2 Off-line program development).

Incorporation of C program blocks

On the PLC, a C program block is called from the standard basic program. The standard basic PLC program contains power-up and initialization routines, establishes the connection to the NCK, machine control panel and operator panels and detects error and operational messages. The basic program can be fully parameterized and controlled with C functions. For a detailed description of the basic program functionality, please refer to

References: /PLCGP/, Description of Functions: Standard Machine

On the PLC, a control program (AB_START.OBJ and ABMAIN.OBJ) is responsible for C block call management and branches into the various runtime levels of the C block depending on which STEP 7 runtime level has issued the call (see Fig. 1-2 Call interface for the C block). The basic PLC program and the call management are not generally changed.

This runtime environment is simulated on the PC by the CS7DLIB library and Borland IDE (see Fig. 1-1 Development of C-PLC programs with Borland C and CS7DLIB). Branching to the various runtime levels can be controlled via the simulation (see Section 3.2 Off-line program development).

Program bodies are available for the possible runtime levels. These serve as call shells for the C application.

A C program file gp840d.c (header: gp840d.h) is supplied and offers access to NCK functionality (see Section C Call Interface for the Basic PLC Program).

The examples supplied for simulation and visualization can be adapted or extended for the off-line test (see Section 3.2 Off-line program development).

Example project

The example project supplied has a standard structure which is tailored to NCK applications. It can be executed immediately in the off-line development environment.

Control file ABMAIN.BAT can be used to generate a C block from the source files of the example project; this C block can be executed immediately on the PLC.

3.2 Off-line program development

What is in this Section?

This Section explains the structure and handling of the CS7DLIB off-line development environment on the basis of the supplied example project *Rotary table*. The example below can be translated and executed immediately both in the PC development environment and in the PLC development environment.

Note

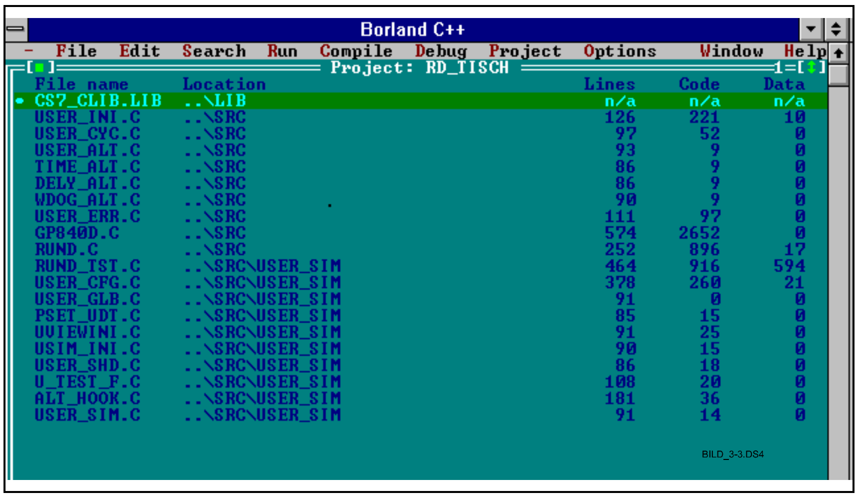
Please refer to Section 5.4 Example project: Rotary table positioning for a description of the example project.

3.2.1 Example project: rotary table control

Open example project

If you want to open the example project, please start the integrated development environment (IDE) for DOS of Borland C++. First select the directory which you specified when setting up the demo project in the Project | Open menu by entering the directory name or selecting the drive and directories. Now go into the subdirectory named project where you will find the project file *rd_tisch.prj*.

Now activate menu item Window | Project. The project window of the example project will then appear on your Borland interface (see Fig. 3-3 Project window of example project).



Borland C++
- File Edit Search Run Compile Debug Project Options Window Help
Project: RD_TISCH
1
BILD_3-3.DS4

```
  File name      Location                          Lines    Code    Data
• CS7_CLIB.LIB   ..\LIB                              n/a     n/a     n/a
  USER_INI.C     ..\SRC                              126     221      10
  USER_CYC.C     ..\SRC                               97      52       0
  USER_ALT.C     ..\SRC                               93       9       0
  TIME_ALT.C     ..\SRC                               86       9       0
  DELY_ALT.C     ..\SRC                               86       9       0
  WDOG_ALT.C     ..\SRC                               90       9       0
  USER_ERR.C     ..\SRC                              111      97       0
  GP840D.C       ..\SRC                              574    2652       0
  RUND.C         ..\SRC                              252     896      17
  RUND_TST.C     ..\SRC\USER_SIM                     464     916     594
  USER_CFG.C     ..\SRC\USER_SIM                     378     260      21
  USER_GLB.C     ..\SRC\USER_SIM                      91       0       0
  PSET_UDT.C     ..\SRC\USER_SIM                      85      15       0
  UVIEWINI.C     ..\SRC\USER_SIM                      91      25       0
  USIM_INI.C     ..\SRC\USER_SIM                      90      15       0
  USER_SHD.C     ..\SRC\USER_SIM                      86      18       0
  U_TEST_F.C     ..\SRC\USER_SIM                     108      20       0
  ALT_HOOK.C     ..\SRC\USER_SIM                     181      36       0
  USER_SIM.C     ..\SRC\USER_SIM                      91      14       0
```

Fig. 3-3 Project window of example project

Files in project window

The project window contains all the files required to generate the example application in an executable form.

Library

The first entry in this project window, i.e. file cs7dlib.lib, is the library supplied with CS7DLIB. It provides you with a runtime environment and PLC functions on your PC as well as special functions for the off-line test.

Source files for runtime levels

The next files entered in the project window, i.e.
user_ini.c, user_cyc.c, *user_alt.c, time_alt.c, dely_alt.c, wdog_alt.c and user_err.c*
contain the function bodies for the actual PLC user program. The user can program his PLC application in these files.

- user_ini.c contains the start function *StdApplStart()* which is called during start-up (see OB100). This function calls the C basic program *basic program startup()* of file gp840d.c (see project window) in this example and initializes the error handler (see Section C Call Interface for the Basic PLC Program).
- user_cyc.c contains the function *StdApplCycle()* which is called up in the free cycle (see OB1). This function calls the C basic program extension *MsttAnNahtstelle()* of file gp840d.c and the rotary table control *rotary table()* of file rund.c in this example project (see project window).
- user_alt.c contains the start function *StdProcessAlert()* which is started by the process interrupted (see OB40).
- time_alt.c contains the function *StdTimeAlert()* which is time-of-day alarm triggered (see OB10).
- dely_alt.c contains the function *StdDelayedTimeAlert()* which is triggered by delay alarms (see OB20).
- wdog_alt.c contains the function *StdWatchdogAlert()* which is triggered by the time alarm (see OB35).
- user_err.c contains the user-specific error handler *StdErrorHandler()* which is called in the event of an erroneous S7 system access operation from the on-line library cs7rtlib.lno (CS7DLIB outputs a message in the status line in the event of an error, but does not call this error handler).

Source files for example project

The following source-code files belong to the example project (demo project); they are not, therefore, an integral part of CS7DLIB and can be modified or omitted.

- gp840d.c contains C functions (see Section C Call Interface for the Basic PLC Program).
- rund.c contains the example program for rotary table positioning (sequence control).

The functions of the files named above are executable immediately, both on the PC and on the PLC (after compilation and integration into the relevant infrastructure).

Source files for simulation

All other files of the project window contain functions for testing, visualization and simulation and can be executed on the PC under CS7DLIB.

Most of these files contain an example code or act as dummies for example codes in order to illustrate the options available with CS7DLIB. Comments are given at the locations at which you can delete the example code or remove it in order to replace it with your own routines. If you create a new project (see Section 2.1 Development environment for PC), then this example code is missing from the start.

The following files are supplied as a source code, but are an integral part of CS7DLIB in terms of their functions and variables. Please enter your own code sequences only at the locations marked by a comment and do not alter either existing variable names, function names or the existing entries in the associated header files.

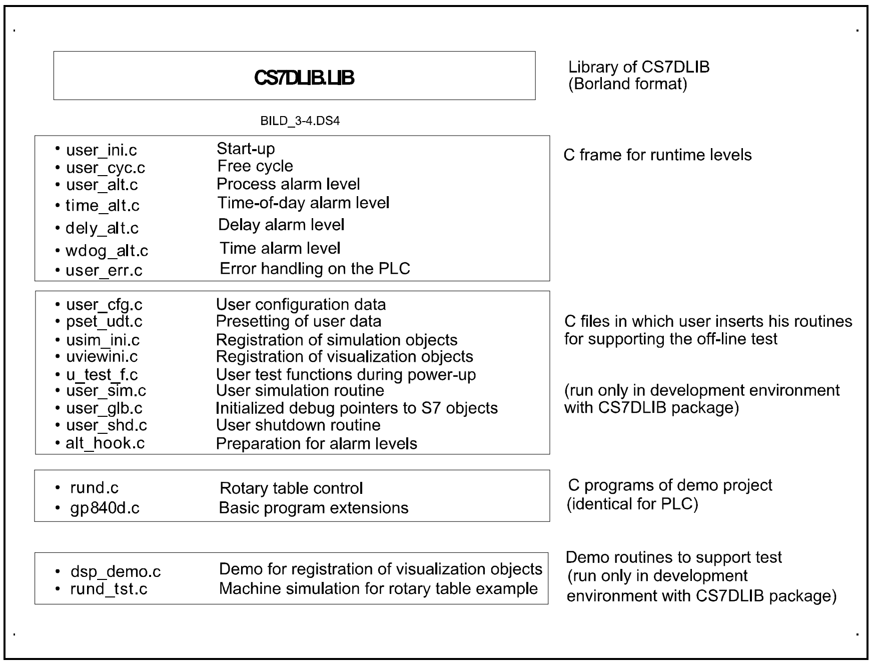
- user_cfg.c is used to set a series of application-specific configuration data for the off-line environment. These data also include the definitions of the data blocks required by the user. The system then converts these configuration entries.
- pset_udt.c is called in the start-up phase of the test system and allows you to preset values for data and S7 objects. When the test system is terminated with the escape key, the status of the S7 objects is stored on restart. This system is restored again when the test system is restarted. This restoration is followed by execution of pset_udt.c.
- usim_ini.c is used to initialize the user simulation. The system activates this function on system power-up.
- uviewini.c provides you with space to insert the registration routines for user-defined visualization objects. The system activates this function on system start-up.
- u_test_f.c provides you with space to insert the calls for user test functions during system start-up. The user can perform preliminary tests on his functions independently of the system in this file; any required "clean-up" operations (e.g. resetting S7 objects and data, deletion of unneeded data blocks) are automatically performed by the system.
- user_sim.c provides you with space to insert user simulation routines which must be called in every cycle.
- user_glb.c contains standardized debug pointers which have been initialized by the system. These allow you to view the current values of the system objects with the aid of the integrated Borland debugger.
- user_shd.c is called when the test system is terminated with the escape key, i.e. in the shutdown phase, and permits clean-up and saving operations to be performed.
- alt_hook.c is used to preprocess the alarm processing operation. Prior to execution of the alarm functions, the test system branches into the appropriate hooks. Here, the user has the opportunity to enter presettings if required.

Source files for simulation of example project

The following source code files belong to the example project (demo project); they are not therefore an integral part of CS7DLIB and can be omitted or replaced by user files. In these files, you will find a range of principles for the application of system facilities. It is therefore worthwhile copying code sequences from these files into your own applications and adapting them to your own requirements:

- rund_tst.c contains an example user program for testing the rotary table positioning function. You will find both user-defined visualization objects as well as the user simulation routine in this file.
- dsp_demo.c demonstrates the initialization and registration of many standard visualization objects. Use this file as a basis for your own visualization objects. The value types to be specified are identified by comments so that you can incorporate your requirements in the same way as in a form.

The following diagram provides you with an overview of the files concerned plus brief comments.



CS7DLIB.LIB		Library of CS7DLIB (Borland format)
BILD_3-4.DS4		
• user_ini.c	Start-up	C frame for runtime levels
• user_cyc.c	Free cycle	
• user_alt.c	Process alarm level	
• time_alt.c	Time-of-day alarm level	
• dely_alt.c	Delay alarm level	
• wdog_alt.c	Time alarm level	
• user_err.c	Error handling on the PLC	
• user_cfg.c	User configuration data	C files in which user inserts his routines for supporting the off-line test
• pset_udt.c	Presetting of user data	
• usim_ini.c	Registration of simulation objects	
• uviewini.c	Registration of visualization objects	
• u_test_f.c	User test functions during power-up	(run only in development environment with CS7DLIB package)
• user_sim.c	User simulation routine	
• user_glb.c	Initialized debug pointers to S7 objects	
• user_shd.c	User shutdown routine	
• alt_hook.c	Preparation for alarm levels	
• rund.c	Rotary table control	C programs of demo project (identical for PLC)
• gp840d.c	Basic program extensions	
• dsp_demo.c	Demo for registration of visualization objects	Demo routines to support test (run only in development environment with CS7DLIB package)
• rund_tst.c	Machine simulation for rotary table example	

Fig. 3-4 Overview of source files of example project

The functions for test support of CS7DLIB, which are available on the PC, but not on the S7 side, can be found in header file cs7_dlib.h.

3.2.2 Header files for PLC and PC environments

In addition to the implementation files (file extension c.) listed in Section 3.2.1 Example project: rotary table control there are other important source files to be noted.

Header files for function prototypes

Each implementation file has a so-called header file with the function prototypes of the implementation file concerned. Function prototypes are the so-called function headers, i.e. they define the call structure of the functions including parameters. The files containing the function prototypes have the same file name as the implementation files. The only difference is that the file extension is .h instead of .c.

If an implementation file now calls a function which is located in another implementation file, then the prototype header file of this other implementation file must be included in the implementation file calling the function by means of the include instruction. Failure to do so will generally lead to compiler warnings and possible to a system crash or malfunctions during processing.

The prototypes of the S7 PLC functions offered by the library are entered in the supplied header file cs7_clib.h . For this reason, the supplied implementation files always include the entry of an include instructions to this header file cs7_clib.h. You should therefore also enter this instruction in the implementation files you create yourself.

Higher-level header files for constants and data types

The prototype header files refer to higher-level header files which generally contain constants as well as the declaration of data types which are relevant for more than one implementation file. This hierarchy allows a clearly structured modularization to be implemented on a large scale, permitting modules to be removed (if they are not required for functional purposes) and to be added relatively easily and without adverse affects.

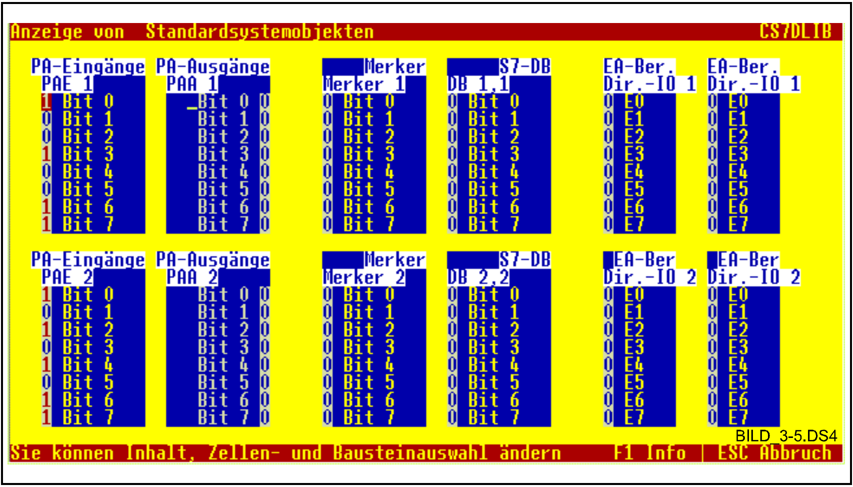
The additions for the test environment are structured according to the same principle. The prototypes of the additional services of library cs7dlib are entered in header file cs7_dlib.h . Implementation files intended solely for the test environment contain therefore the include instruction for the two library header files cs7_clib.h and cs7_dlib.h in addition to the instruction for their own prototype header files.

System header files

System header files are stored under the path ..\sys_inc; user-defined header files should be stored under the path ..\user_inc.

3.2.3 Standard visualization objects

CSDLIB offers a range of standard display objects which can be called by means of function keys. These objects were specially selected to ensure that several representatives of each supported S7 object are available so that the user is provided quickly with basic, ready-programmed tools for testing an application. With these tools, he can monitor and influence the effects of important sections of an application. CS7DLIB offers visualization objects for the S7 objects.



```
Anzeige von  Standardsystemobjekten                                 CS7DLIB

 PA-Eingänge PA-Ausgänge      Merker     S7-DB      EA-Ber.    EA-Ber.
  PAE 1       PAA 1       Merker 1   DB 1,1     Dir.-IO 1  Dir.-IO 1
  1 Bit 0       _Bit 0 0  0 Bit 0    0 Bit 0    0 E0       0 E0
  0 Bit 1        Bit 1 0  0 Bit 1    0 Bit 1    0 E1       0 E1
  0 Bit 2        Bit 2 0  0 Bit 2    0 Bit 2    0 E2       0 E2
  1 Bit 3        Bit 3 0  0 Bit 3    0 Bit 3    0 E3       0 E3
  0 Bit 4        Bit 4 0  0 Bit 4    0 Bit 4    0 E4       0 E4
  0 Bit 5        Bit 5 0  0 Bit 5    0 Bit 5    0 E5       0 E5
  1 Bit 6        Bit 6 0  0 Bit 6    0 Bit 6    0 E6       0 E6
  1 Bit 7        Bit 7 0  0 Bit 7    0 Bit 7    0 E7       0 E7

 PA-Eingänge PA-Ausgänge      Merker     S7-DB      EA-Ber     EA-Ber
  PAE 2       PAA 2       Merker 2   DB 2,2     Dir.-IO 2  Dir.-IO 2
  1 Bit 0        Bit 0 0  0 Bit 0    0 Bit 0    0 E0       0 E0
  0 Bit 1        Bit 1 0  0 Bit 1    0 Bit 1    0 E1       0 E1
  1 Bit 2        Bit 2 0  0 Bit 2    0 Bit 2    0 E2       0 E2
  0 Bit 3        Bit 3 0  0 Bit 3    0 Bit 3    0 E3       0 E3
  1 Bit 4        Bit 4 0  0 Bit 4    0 Bit 4    0 E4       0 E4
  0 Bit 5        Bit 5 0  0 Bit 5    0 Bit 5    0 E5       0 E5
  1 Bit 6        Bit 6 0  0 Bit 6    0 Bit 6    0 E6       0 E6
  1 Bit 7        Bit 7 0  0 Bit 7    0 Bit 7    0 E7       0 E7
                                                              BILD_3-5.DS4
Sie können Inhalt, Zellen- und Bausteinauswahl ändern    F1 Info | ESC Abbruch
```

Fig. 3-5 Display of standard visualization objects

The system automatically displays 4 screens with different combinations.

When you can see the bit assignment in the bit display, you will find it is possible to interpret the values quickly and to change them just as quickly by entering new ones. The changed bits remain valid over and beyond the life of the program if you terminate the program with the ESC key and the value is not overwritten by the program.

The set object display remains unchanged during the program life provided it is not altered. Even if you switch over to other screens, you will still find the set object display on the original screen when you return to it later.

Operator control

Keys have been defined for operator control which are based on commonly used key assignments inasmuch as comparable examples exist. A few notes on operator control are given below:

- When the demo program is started, screen displays can be called via function keys. Some of these displays are a feature of the system, others originate from the user program.
- If S7 objects are visualized, then the following keys can be used to control them:
  – Spacebar:
    Inverts the bit on which the red marker is positioned
  – Cursor key down:
    Selects the following signal of the object
  – Cursor key up:
    Selects the preceding signal of the object

- Page down key:
  Selects the last signal of the object
- Page up key:
  Selects the first signal of the object
- CTRL key and page down key together:
  Selects the last signal of the last object
- CTRL key and page up key together:
  Selects the first signal of the first object
- Tab key:
  Selects the next object
- Shift and Tab keys together:
  The preceding object is selected
- CTRL key and cursor to right keys together
  Increments the number of the S7 object
  (e.g. PII 1 -> PII 2)
- CTRL key and cursor to left key together
  Decrements the number of the S7 object
  (e.g. memory bit 15 -> memory bit 14)
- Pos1 key:
  Selects S7 object with the number 0
  (e.g memory bit 15 -> memory bit 0)
- End key:
  Selects S7 object with the number 127
  (e.g. memory bit 15 -> memory bit 127)
- CTRL key and Pos1 key together:
  Selects the S7 object with a number which is 16 lower
  (e.g. memory bit 50 -> memory bit 34)
- CTRL key and End key together:
  Selects the S7 object with a number which is 16 higher
  (e.g. memory bit 50 -> memory bit 66)
- Plus key (+)
  Increments the data byte for a data block
  (e.g. DB 20 byte 3 -> DB 20 byte 4)
- Minus key (-)
  Decrements the data byte for a data block
  (e.g. DB 20 byte 3 -> DB 20 byte 2)
- The program is aborted with the ESC key. In this case, altered bits are stored in a file (file name can be specified in the supplied C source file user_cfg.c). These bits are restored on the next start and become visible again provided they have not been overwritten by the user.

The most important keys are summarized in Fig. 3-6 Operator control of visualization objects.

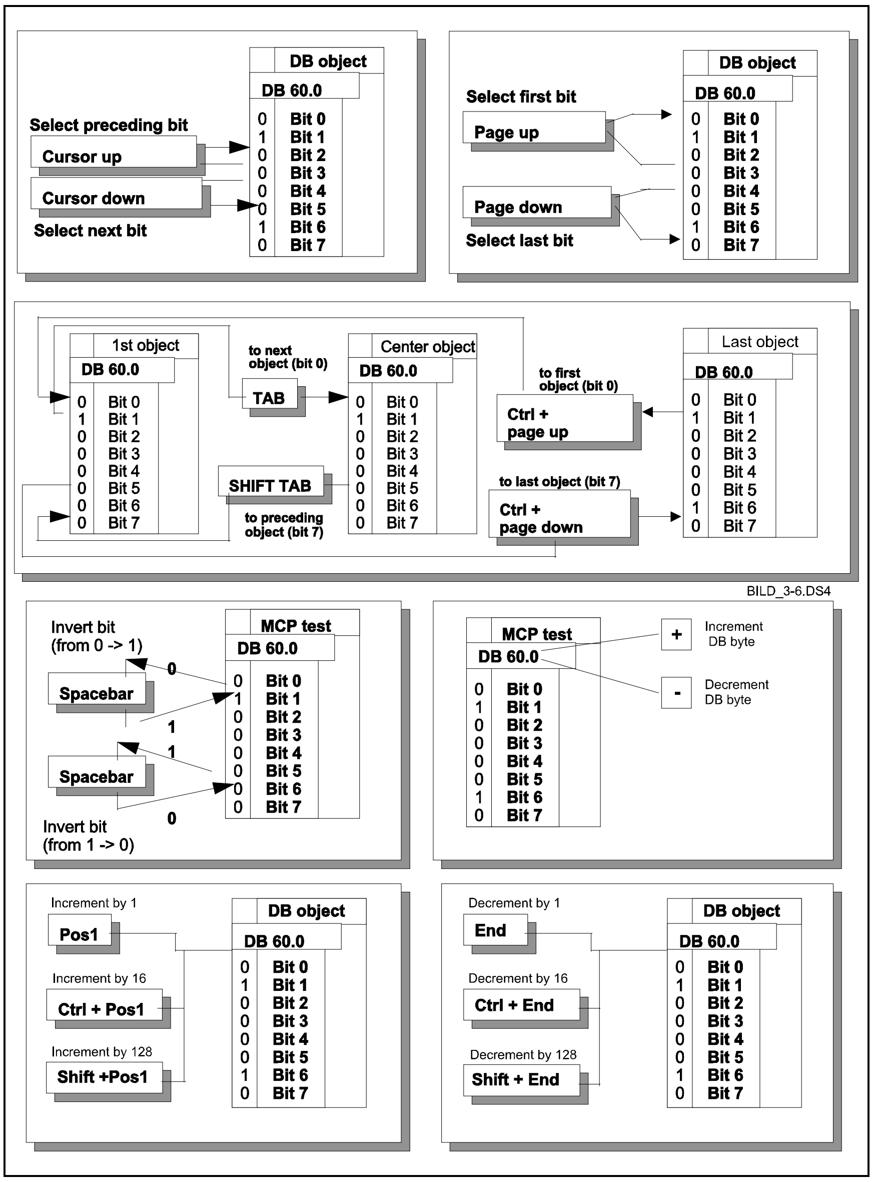


Fig. 3-6 Operator control of visualization objects

3.2.4 User-defined visualization objects

There are certain typical displays with a particular meaning which recur frequently in projects. In this case, it is worth the extra effort of assigning texts once to the signals and registering corresponding, symbolic displays. You can do this quickly and easily by using the template in the example file dsp_demo.c. Copy the relevant template into your application and change it. In this way, you can create a means for controlling and monitoring your application which you can activate and switch over quickly.

Integrating visualization objects

The example file dsp_demo.c shows you how you can integrate visualization objects to suit your own needs. Experiment with these objects by making changes in the example project and then copy them to your own application and change them there.

Fig. 3-7 Visualization facilities and Fig. 3-8 Identifiers for visualization objects give a summary of the visualization facilities available.

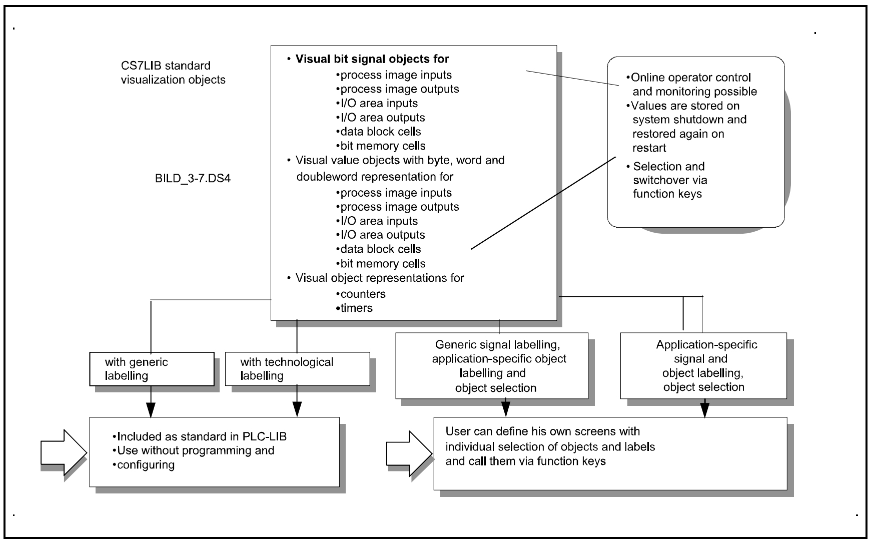

CS7LIB standard visualization objects
BILD_3-7.DS4
• Visual bit signal objects for
•process image inputs
•process image outputs
•I/O area inputs
•I/O area outputs
•data block cells
•bit memory cells
• Visual value objects with byte, word and doubleword representation for
•process image inputs
•process image outputs
•I/O area inputs
•I/O area outputs
•data block cells
•bit memory cells
• Visual object representations for
•counters
•timers
•Online operator control and monitoring possible
•Values are stored on system shutdown and restored again on restart
• Selection and switchover via function keys
with generic labelling
with technological labelling
Generic signal labelling, application-specific object labelling and object selection
Application-specific signal and object labelling, object selection
•Included as standard in PLC-LIB
•Use without programming and
•configuring
User can define his own screens with individual selection of objects and labels and call them via function keys

Fig. 3-7 Visualization facilities

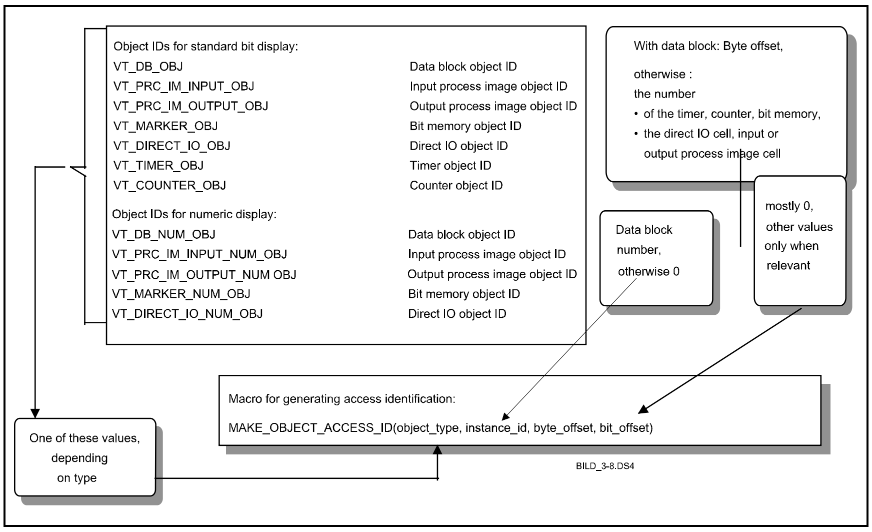
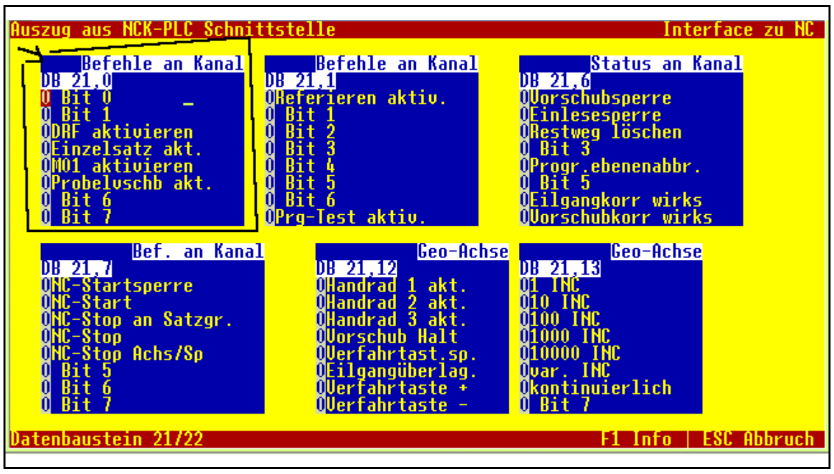

Object IDs for standard bit display:
VT_DB_OBJ Data block object ID
VT_PRC_IM_INPUT_OBJ Input process image object ID
VT_PRC_IM_OUTPUT_OBJ Output process image object ID
VT_MARKER_OBJ Bit memory object ID
VT_DIRECT_IO_OBJ Direct IO object ID
VT_TIMER_OBJ Timer object ID
VT_COUNTER_OBJ Counter object ID
Object IDs for numeric display:
VT_DB_NUM_OBJ Data block object ID
VT_PRC_IM_INPUT_NUM_OBJ Input process image object ID
VT_PRC_IM_OUTPUT_NUM OBJ Output process image object ID
VT_MARKER_NUM_OBJ Bit memory object ID
VT_DIRECT_IO_NUM_OBJ Direct IO object ID
With data block: Byte offset, otherwise : the number • of the timer, counter, bit memory, • the direct IO cell, input or output process image cell
Data block number, otherwise 0
mostly 0, other values only when relevant
Macro for generating access identification:
MAKE_OBJECT_ACCESS_ID(object_type, instance_id, byte_offset, bit_offset)
One of these values, depending on type
BILD_3-8.DS4

Fig. 3-8 Identifiers for visualization objects

You can create screen pages yourself using S7 objects. In this case, you can select the position of each object, assign a logical name to each object and determine the designations of the individual signals of the objects by entering your own texts (example: See Fig. 3-9 Example of a screen page).



```
Auszug aus NCK-PLC Schnittstelle                              Interface zu NC

      Befehle an Kanal      Befehle an Kanal           Status an Kanal
DB 21,0                DB 21,1                 DB 21,6
0 Bit 0                0Referieren aktiv.      0Vorschubsperre
0 Bit 1                0 Bit 1                 0Einlesesperre
0DRF aktivieren        0 Bit 2                 0Restweg löschen
0Einzelsatz akt.       0 Bit 3                 0 Bit 3
0M01 aktivieren        0 Bit 4                 0Progr.ebenenabbr.
0Probelvschb akt.      0 Bit 5                 0 Bit 5
0 Bit 6                0 Bit 6                 0Eilgangkorr wirks
0 Bit 7                0Prg-Test aktiv.        0Vorschubkorr wirks

         Bef. an Kanal           Geo-Achse          Geo-Achse
DB 21,7                  DB 21,12             DB 21,13
0NC-Startsperre          0Handrad 1 akt.      01 INC
0NC-Start                0Handrad 2 akt.      010 INC
0NC-Stop an Satzgr.      0Handrad 3 akt.      0100 INC
0NC-Stop                 0Vorschub Halt       01000 INC
0NC-Stop Achs/Sp         0Verfahrtast.sp.     010000 INC
0 Bit 5                  0Eilgangüberlag.     0var. INC
0 Bit 6                  0Verfahrtaste +      0kontinuierlich
0 Bit 7                  0Verfahrtaste -      0 Bit 7
Datenbaustein 21/22                                   F1 Info | ESC Abbruch
```

Fig. 3-9 Example of a screen page

Some of the components of the screen page are, for example, a left-aligned and right-aligned text in the title bar and a left-aligned text in the screen footer. The S7 objects are represented in the form of rectangles, each of which has a user-defined object designation justified to the right in the 1st line. The abbreviated, current S7 object designation is located in the 2nd line of each rectangle.

Example:
NCK-PLC interface of SINUMERIK 840D

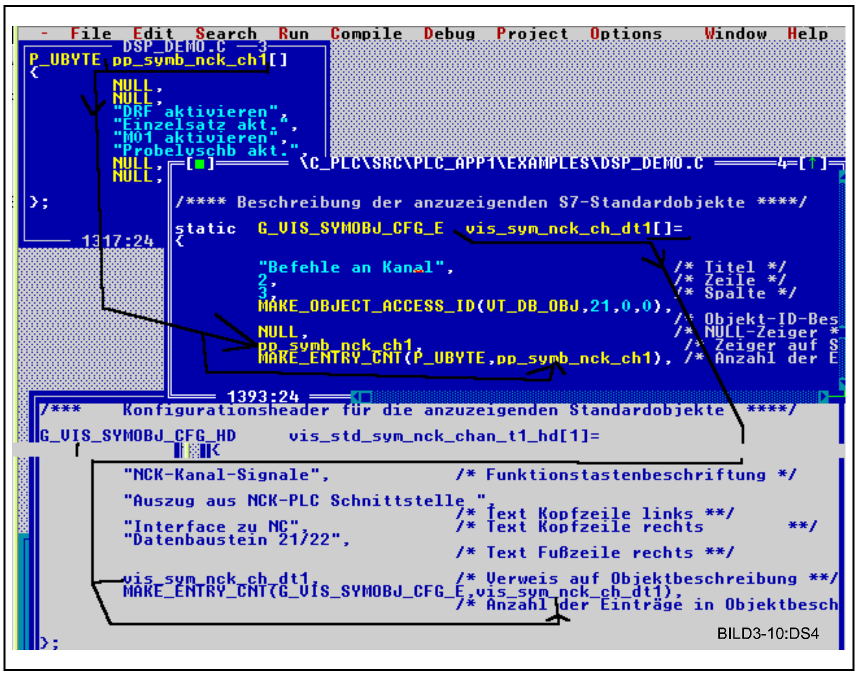
The example given here shows an extract of the NCK-PLC interface of the SINUMERIK 840D.

The symbolic designator is DB 20,0; in this case, "DB" stands for data block, "20" for data block number 20 and "0" for byte 0 of data block 20. Since the symbolic signals apply only to this byte, it is not possible to page on to other bytes in this data block nor is it possible to change the data block through an operator input.

Every bit is marked by its current state, i.e. 0 or 1, and also has a text which has been assigned by the user. Non-assigned bits can be identified by the text "Bit x", where "x" represents the bit number.

Using the cursor keys, you can position the cursor on the bit of your choice. Now press the spacebar and the bit will invert its value, i.e. change from 0 to 1 or from 1 to 0. Provided the signal is not overwritten by the running program, then it retains this value. If you exit the system in the normal way, i.e. with the ESC key, then the current signal value is stored and restored again when the application is restarted.

Fig. 3-10 and Fig. 3-11 Generating screen pages below show you how you can generate this type of screen page:


- File Edit Search Run Compile Debug Project Options Window Help
DSP_DEMO.C 3
1317:24
\C_PLC\SRC\PLC_APP1\EXAMPLES\DSP_DEMO.C 4
1393:24
BILD3-10:DS4

```
P_UBYTE pp_symb_nck_ch1[]
{
        NULL,
        NULL,
        "DRF aktivieren",
        "Einzelsatz akt.",
        "M01 aktivieren",
        "Probelvschb akt.",
        NULL,
        NULL,

};
```

```
/**** Beschreibung der anzuzeigenden S7-Standardobjekte ****/

static  G_VIS_SYMOBJ_CFG_E  vis_sym_nck_ch_dt1[]=
{

        "Befehle an Kanal",                              /* Titel */
        2,                                               /* Zeile */
        3,                                               /* Spalte */
        MAKE_OBJECT_ACCESS_ID(VT_DB_OBJ,21,0,0),         /* Objekt-ID-Bes
        NULL,                                            /* NULL-Zeiger *
        pp_symb_nck_ch1,                                  /* Zeiger auf S
        MAKE_ENTRY_CNT(P_UBYTE,pp_symb_nck_ch1),         /* Anzahl der E
```

```
/***    Konfigurationsheader für die anzuzeigenden Standardobjekte  ****/

G_VIS_SYMOBJ_CFG_HD     vis_std_sym_nck_chan_t1_hd[1]=
{

        "NCK-Kanal-Signale",                 /* Funktionstastenbeschriftung */

        "Auszug aus NCK-PLC Schnittstelle ",
                                             /* Text Kopfzeile links **/
        "Interface zu NC",                   /* Text Kopfzeile rechts      **/
        "Datenbaustein 21/22",
                                             /* Text Fußzeile rechts **/

        vis_sym_nck_ch_dt1,                  /* Verweis auf Objektbeschreibung **/
        MAKE_ENTRY_CNT(G_VIS_SYMOBJ_CFG_E,vis_sym_nck_ch_dt1),
                                             /* Anzahl der Einträge in Objektbesch

};
```

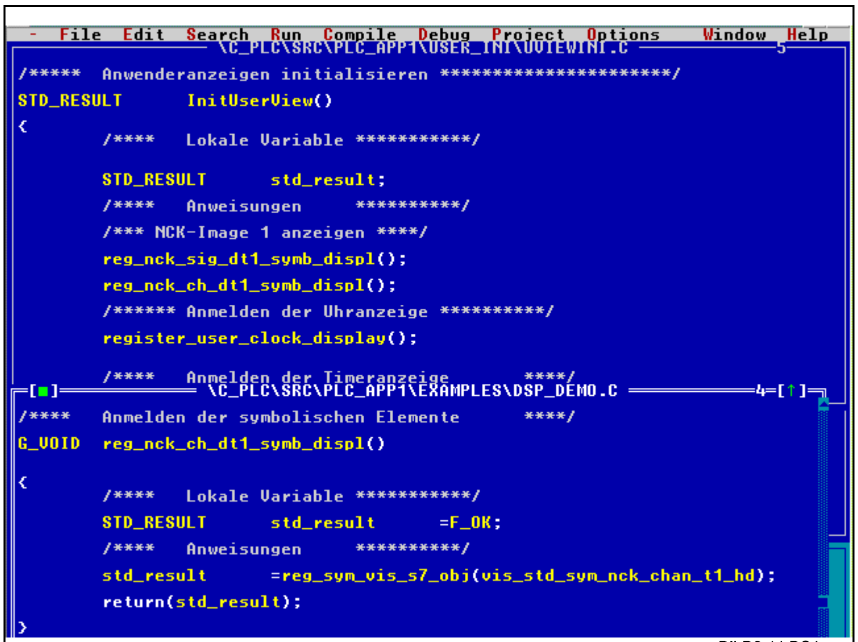
Fig. 3-10 Generating screen pages

For the purpose of understanding the process, let us concentrate on the first element on the screen because you use the same procedure for all the other elements.

First of all, write down the text for the bits (see upper left-hand window) and embed them in an object to which you should assign a name. Make sure that the definition of your object is correct. The object represents an array of pointers to bytes. It is best if you copy the example object with the editor and then enter your own object designation and texts. This is the easiest and least complicated method and helps to avoid a lot of errors right from the start.

After you have done this for each of the objects to be displayed on the screen page, generate the next object (preferably by copying and altering the example template) which contains all the objects on the screen page. Remember to specify the following for each individual object: Its title (first line in object display), the line and column on the screen in which it must be displayed, which byte of which S7 object it represents and finally, a reference to the texts of each bit for this S7 object.

The next step is to define the screen. Here you need to specify the text with which you want to call it. This text is entered in the function key overview (call by pressing function key F1) and acts as the identifier. You must then specify the texts which are output on the left in the header and footer. Now you need to enter the reference to the general object descriptions (i.e. those which include all individual objects) and the number of objects. To save you having to count yourself, you will be assisted here by a macro which also registers new entries.


- File Edit Search Run Compile Debug Project Options Window Help
\C_PLC\SRC\PLC_APP1\USER_INI\UVIEWINI.C 5
[■] \C_PLC\SRC\PLC_APP1\EXAMPLES\DSP_DEMO.C 4=[↑]

```
/*****  Anwenderanzeigen initialisieren **********************/
STD_RESULT      InitUserView()
{
        /****   Lokale Variable ***********/

        STD_RESULT      std_result;
        /****   Anweisungen     **********/
        /*** NCK-Image 1 anzeigen ****/
        reg_nck_sig_dt1_symb_displ();
        reg_nck_ch_dt1_symb_displ();
        /****** Anmelden der Uhranzeige **********/
        register_user_clock_display();

        /****   Anmelden der Timeranzeige       ****/
```

```
/****   Anmelden der symbolischen Elemente      ****/
G_VOID  reg_nck_ch_dt1_symb_displ()

{
        /****   Lokale Variable ***********/
        STD_RESULT      std_result      =F_OK;
        /****   Anweisungen     **********/
        std_result      =reg_sym_vis_s7_obj(vis_std_sym_nck_chan_t1_hd);
        return(std_result);
}
```

BILD3-11.DS4

Fig. 3-11 Generating screen pages

Assignment of function keys

You can always view the assignment of the function keys by pressing function key F1 when an application is active. The first keys are allocated to predefined system displays. When you register your visualization objects, you occupy further function keys. After function key F10, you must press the CTRL key and the function key simultaneously (identified by ^F1, i.e. press CTRL key and + F1 key simultaneously).

3.2.5 Simulation routines

Simulation in example project

CS7DLIB offers you the means to create your own simulation routines. In the example file rund_tst.c of the example project, you will find a machine simulation for sequence control rund.c. as a guide to how this type of simulation can be incorporated in your application.

Experiment with this option by making changes in the example project and then use the facility if you need to.

Fig. 3-12 Simulation with CS7DLIB gives you an overview of the simulation routine option:

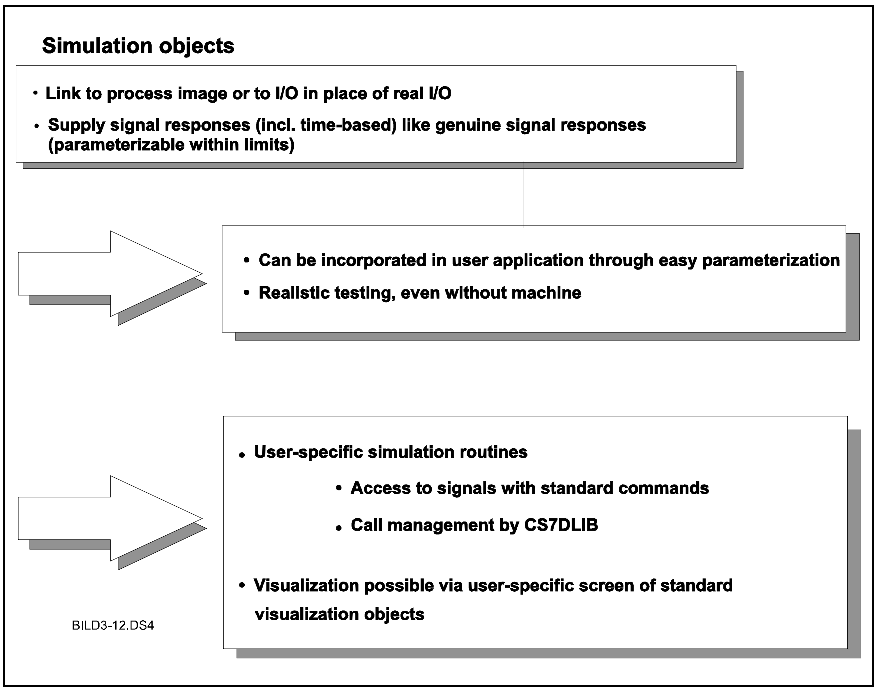

Simulation objects
• Link to process image or to I/O in place of real I/O
• Supply signal responses (incl. time-based) like genuine signal responses (parameterizable within limits)
• Can be incorporated in user application through easy parameterization
• Realistic testing, even without machine
• User-specific simulation routines
• Access to signals with standard commands
• Call management by CS7DLIB
• Visualization possible via user-specific screen of standard visualization objects
BILD3-12.DS4

Fig. 3-12 Simulation with CS7DLIB

3.2.6 Presetting data during start-up of test environment

In order to establish a defined start environment or for the purpose of testing, you can preset data during start-up with CS7DLIB. 'You need to insert the functions for this option in file pset_udt.c which is called by the CS7DLIB during start-up. Please note the information about general initialization in the following diagram:

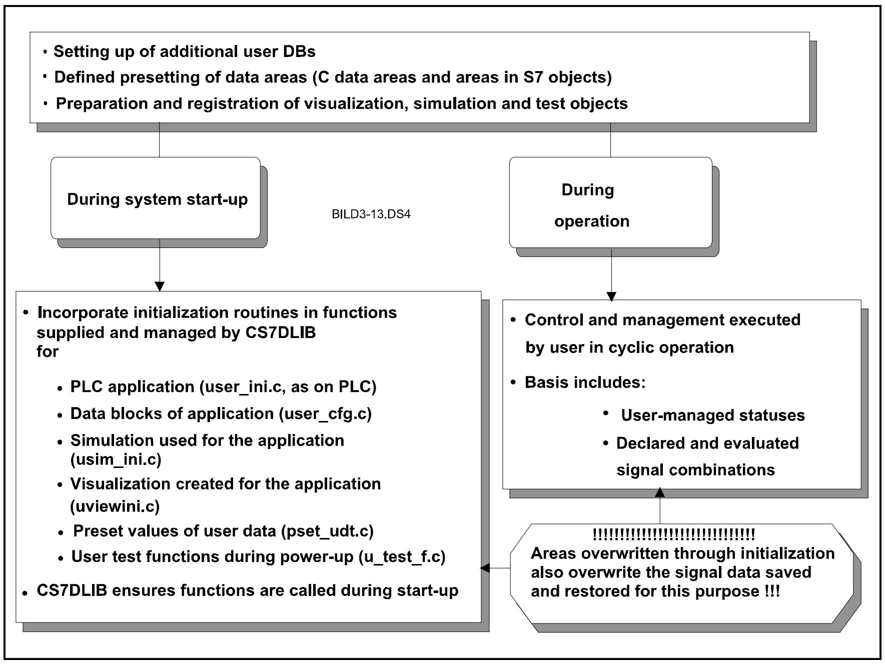

• Setting up of additional user DBs
• Defined presetting of data areas (C data areas and areas in S7 objects)
• Preparation and registration of visualization, simulation and test objects
During system start-up
BILD3-13.DS4
During operation
• Incorporate initialization routines in functions supplied and managed by CS7DLIB for
• PLC application (user_ini.c, as on PLC)
• Data blocks of application (user_cfg.c)
• Simulation used for the application (usim_ini.c)
• Visualization created for the application (uviewini.c)
• Preset values of user data (pset_udt.c)
• User test functions during power-up (u_test_f.c)
• CS7DLIB ensures functions are called during start-up
• Control and management executed by user in cyclic operation
• Basis includes:
• User-managed statuses
• Declared and evaluated signal combinations
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Areas overwritten through initialization also overwrite the signal data saved and restored for this purpose !!!

Fig. 3-13 Initialization of off-line test environment

3.2.7 Termination procedure

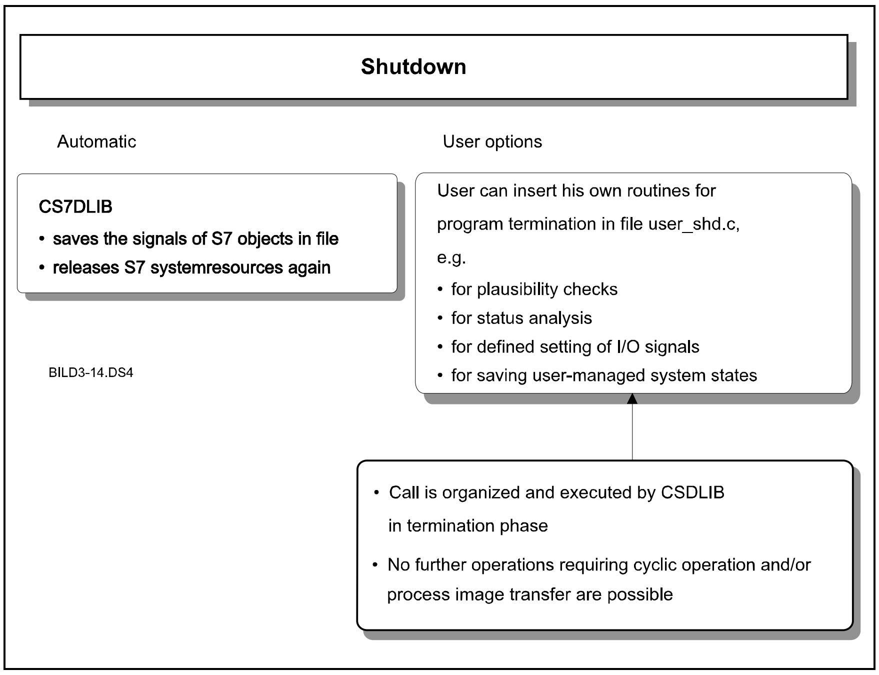

Shutdown
Automatic
User options
CS7DLIB
• saves the signals of S7 objects in file
• releases S7 systemresources again
BILD3-14.DS4
User can insert his own routines for program termination in file user_shd.c, e.g.
• for plausibility checks
• for status analysis
• for defined setting of I/O signals
• for saving user-managed system states
• Call is organized and executed by CSDLIB in termination phase
• No further operations requiring cyclic operation and/or process image transfer are possible

Fig. 3-14 Overview of program termination routines

It is particularly advisable in test operation to examine and save certain data and states when the program is terminated. You can insert the functions required for this purpose in file user_shd.c. CS7DLIB calls this file in the termination phase. An analysis procedure using debug points may be included in these routines if required.

3.2.8 Setting configuration data

CS7DLIB provides the user with a number of configuring options. To make use of these, it is necessary to make appropriate entries or modifications in the project-specific file user_cfg.c. The available options and supplementary conditions are documented in this file by means of comments. Please observe the specified supplementary conditions to avoid any malfunctions later.

You move around in file user_cfg.c in a similar way as you do in a form except that in this case, the frame is specified by C statements and the file is interpreted by the compiler and the CS7DLIB.

Setting options

The following setting options are currently available:

- Input of file in which the S7 object contents are saved. It is possible to implement various scenarios through modification and to return to predefined scenarios when required.
- Input of user data blocks with number and size. An entry block must always be used or else the computer will signal an error.
- Input of machine configuration (number of mode groups, channels, axes)
- List in which the user can optionally control the transfer of data between the simulated I/O area and the process image. This list must contain at least one entry block or else the computer will signal an error.

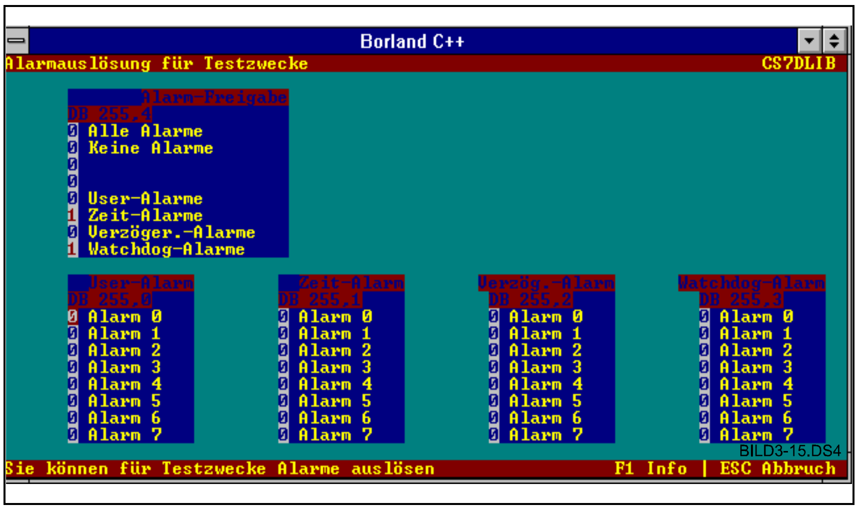
3.2.9 Test of alarm runtime levels

Alarm levels

In addition to the cyclical program section StdApplCycl() and the start-up section StdApplStart(), the following alarm levels exist:

- StdProcessAlert()
- StdTimeAlert()
- StdWatchdogAlert()
- StdDelayedTimeAlert()

These levels can be programmed via SFCs (see Section 3.1.2 Use of data types) or are initiated via system events. Using a preprogrammed S7 display object, you can simulate this type of process using CS7DLIB.



```
Borland C++
Alarmauslösung für Testzwecke                                        CS7DLIB

        Alarm-Freigabe
     DB 255.4
     0 Alle Alarme
     0 Keine Alarme
     0
     0
     0 User-Alarme
     1 Zeit-Alarme
     0 Verzöger.-Alarme
     1 Watchdog-Alarme

        User-Alarm        Zeit-Alarm      Verzög.-Alarm     Watchdog-Alarm
     DB 255.0          DB 255.1           DB 255.2           DB 255.3
     0 Alarm 0         0 Alarm 0          0 Alarm 0          0 Alarm 0
     0 Alarm 1         0 Alarm 1          0 Alarm 1          0 Alarm 1
     0 Alarm 2         0 Alarm 2          0 Alarm 2          0 Alarm 2
     0 Alarm 3         0 Alarm 3          0 Alarm 3          0 Alarm 3
     0 Alarm 4         0 Alarm 4          0 Alarm 4          0 Alarm 4
     0 Alarm 5         0 Alarm 5          0 Alarm 5          0 Alarm 5
     0 Alarm 6         0 Alarm 6          0 Alarm 6          0 Alarm 6
     0 Alarm 7         0 Alarm 7          0 Alarm 7          0 Alarm 7
Sie können für Testzwecke Alarme auslösen              F1 Info | ESC Abbruch
```

BILD3-15.DS4

Fig. 3-15 Visualization objects for alarm levels (standard display)

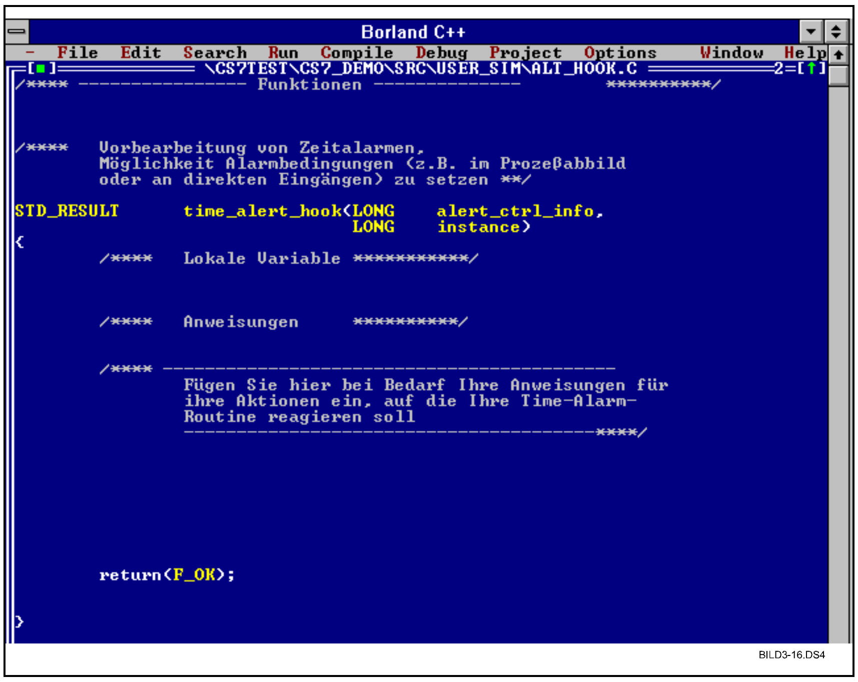
Initiating alarms

On the basis of these objects, you can now initiate 8 different events per alarm level. The respective event is added to the appropriate hook function in the alt_hook.c file for preparation of the alarm simulation. After the hook function, CS7DLIB branches into the actual alarm processing level.

Example for process interrupt events

The example below shows how you can simulate the occurrence of different process interrupt events such as, for example, M40, M19=1, etc. using *alt_hook.c*.

The hook functions receive the transfer parameters *alert_ctrl_info (*event information) and *instance (*containing alarm ID (ID for process interrupt, watchdog alarm, ...))


Borland C++
File Edit Search Run Compile Debug Project Options Window Help
\CS7TEST\CS7_DEMO\SRC\USER_SIM\ALT_HOOK.C
2
BILD3-16.DS4

```
/**** ---------------- Funktionen --------------           **********/


/****   Vorbearbeitung von Zeitalarmen,
        Möglichkeit Alarmbedingungen (z.B. im Prozeßabbild
        oder an direkten Eingängen) zu setzen **/

STD_RESULT      time_alert_hook(LONG    alert_ctrl_info,
                                LONG    instance)
{
        /****   Lokale Variable **********/


        /****   Anweisungen     **********/


        /**** -------------------------------------------
                Fügen Sie hier bei Bedarf Ihre Anweisungen für
                ihre Aktionen ein, auf die Ihre Time-Alarm-
                Routine reagieren soll
                ----------------------------------------****/





        return(F_OK);

}
```

Fig. 3-16 Listing of file alt_hook.c (extract)

3.2.10 Notes on testing

Debugger functions

Borland C++ 3.x (or Turbo C 3.0) contains a user-friendly, integrated debugger with the following functions:

- Move step by step through program (*Single Step*), skipping functions as required or stepping to functions,
- Allow program to run on until a specified point (*break point*) is reached,
- View variables and structures (*Inspect function*),
- Monitor variables and structures continuously (*Watch function*).

Note

Refer to the Borland documentation for more information about handling and functional scope of this tool.

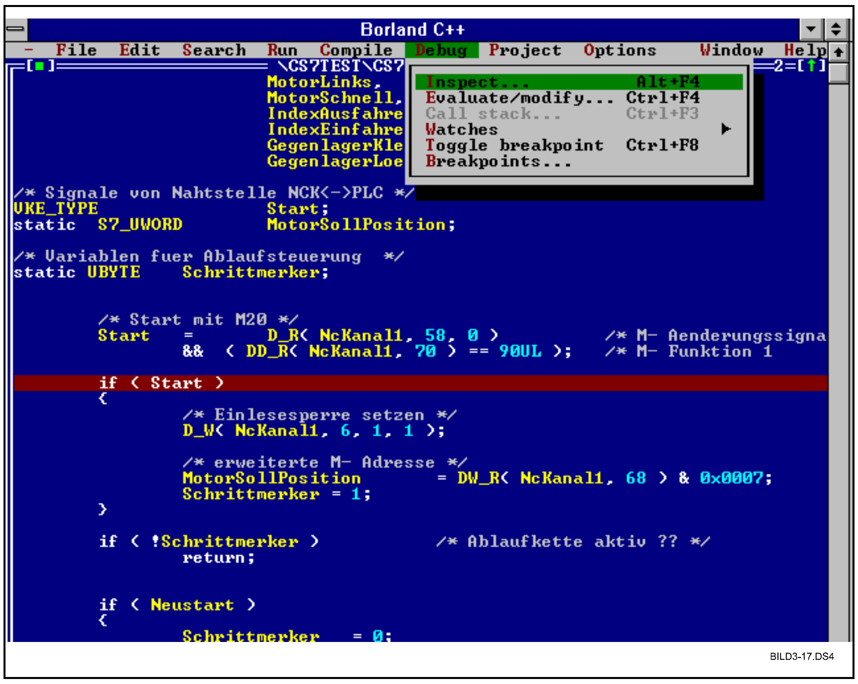
Monitor status and contents of S7 system objects

If you want to monitor the status and contents of S7 system objects with the debugger, then you should use file *user_glb.c*. This file contains pointers which are managed and initialized by the library and which have debugger-friendly data types. The inspect function (can also be moved to the right-hand mouse key) supplies the current values of the clicked object. It may be necessary to open further inspect windows for detailed views by pressing the Return key. You will see any changes when stepping through program sections which modify the contents being monitored of these objects. The use of S7 objects will be explained in the following.

Notes on debugging the example project

Some information about debugging with CS7DLIB based on the example project "Rotary table control" is given below:

- Open the example project. Set a breakpoint in the file RUND.C by pressing CTRL-F8 (toggle breakpoint) in the program line with the sequence *if (start)* (in function *VOID Rotary Table (VOID)*). Then start the program with CTRL-F8.
- After the program has been started, it will be interrupted at this line, the line being highlighted by a coloured background. You can now analyse execution of the program using the debug functions (Single Step, Watch, Inspect).

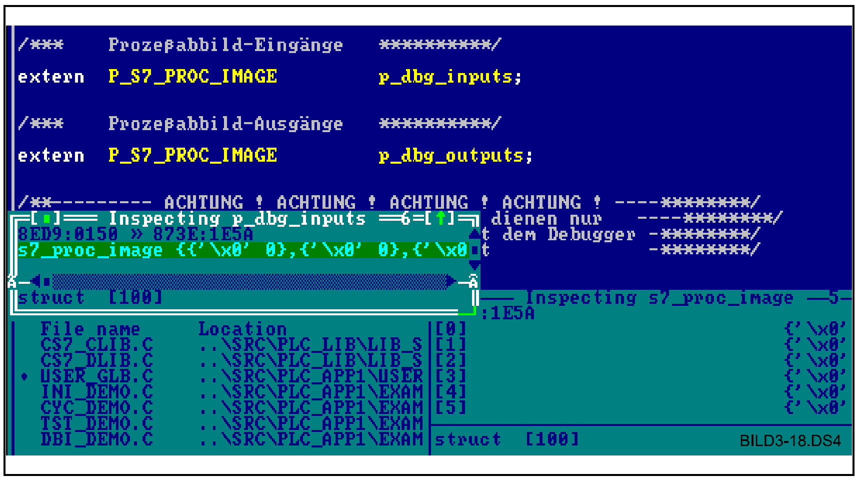

Borland C++
File Edit Search Run Compile Debug Project Options Window Help
\CS7TEST\CS7
Inspect... Alt+F4
Evaluate/modify... Ctrl+F4
Call stack... Ctrl+F3
Watches
Toggle breakpoint Ctrl+F8
Breakpoints...
BILD3-17.DS4

```
                                MotorLinks,
                                MotorSchnell,
                                IndexAusfahre
                                IndexEinfahre
                                GegenlagerKle
                                GegenlagerLoe

/* Signale von Nahtstelle NCK<->PLC */
UKE_TYPE                        Start;
static  S7_UWORD                MotorSollPosition;

/* Variablen fuer Ablaufsteuerung  */
static UBYTE    Schrittmerker;


        /* Start mit M20 */
        Start    =      D_R( NcKanal1, 58, 0 )         /* M- Aenderungssigna
                && ( DD_R( NcKanal1, 70 ) == 90UL );   /* M- Funktion 1

        if ( Start )
        {
                /* Einlesesperre setzen */
                D_W( NcKanal1, 6, 1, 1 );

                /* erweiterte M- Adresse */
                MotorSollPosition        = DW_R( NcKanal1, 68 ) & 0x0007;
                Schrittmerker = 1;
        }

        if ( !Schrittmerker )              /* Ablaufkette aktiv ?? */
                return;


        if ( Neustart )
        {
                Schrittmerker    = 0;
```

Fig. 3-17 Inspect function

Example:
Display process image

- Select file *user_glb.c* in the project window and open the associated editor window by pressing the Return key.
- In the editor window, position the cursor on the C pointer *p_dbg_inputs* which contains the reference to the area of the process input image.
- Open the associated Inspect window by selecting menu item Debug | Inspect.
- To obtain a clearer display of the process input image, mark the line with the symbol *s7_proc_image* in the window you have just opened and press the Return key. This causes a further Inspect window to be opened in which the byte index (enclosed in square brackets) of the process input image is displayed line by line in the left-hand column and the current value of the assigned position is output on the right in two representation modes, i.e. character and decimal representation.
- If you want the program to continue after completing your analysis, press CTRL-F9 (RUN) again. Program execution will now continue provided that no further breakpoints are encountered or a termination command given with ESC.



```
/***     Prozeßabbild-Eingänge     ***********/
extern   P_S7_PROC_IMAGE           p_dbg_inputs;

/***     Prozeßabbild-Ausgänge     ***********/
extern   P_S7_PROC_IMAGE           p_dbg_outputs;

/**---------- ACHTUNG ! ACHTUNG ! ACHTUNG ! ACHTUNG ! ----*********/
╔═[■]══ Inspecting p_dbg_inputs ══6═[↑]═╗ dienen nur    ----*********/
8ED9:0150 » 873E:1E5A                    t dem Debugger -*********/
s7_proc_image {{'\x0' 0},{'\x0' 0},{'\x0 t              -*********/

struct  [100]                            ─── Inspecting s7_proc_image ──5─
                                         :1E5A
   File name      Location              [0]                          {'\x0'
   CS7_CLIB.C     ..\SRC\PLC_LIB\LIB_S  [1]                          {'\x0'
   CS7_DLIB.C     ..\SRC\PLC_LIB\LIB_S  [2]                          {'\x0'
 • USER_GLB.C     ..\SRC\PLC_APP1\USER  [3]                          {'\x0'
   INI_DEMO.C     ..\SRC\PLC_APP1\EXAM  [4]                          {'\x0'
   CYC_DEMO.C     ..\SRC\PLC_APP1\EXAM  [5]                          {'\x0'
   TST_DEMO.C     ..\SRC\PLC_APP1\EXAM
   DBI_DEMO.C     ..\SRC\PLC_APP1\EXAM  struct   [100]
```

Bild 3-18 Process image display

This example shows you how you can process the code step by step and, while doing so, monitor its effects using the Borland debugger tools and the pointers prepared for access to system objects such as process image, counters, timers, bit memories or data blocks (see file *user_glb.c*).

Displaying counters and timers

Counters and timers are not often easy to test because the counter or timer events usually do not occur until the program has been executed many times. In order to test the elementary effect quickly and easily, it is advisable to conduct preliminary tests in the course of system startup. CS7DLIB provides space for these tests in file *u_test_f.c* .

The events or runs can be generated by loops which can easily be examined by means of carefully selected breakpoints in the relevant locations. For testing of timer functions, CS7DLIB also offers the possibility of simulating timer clocks (see *CS7_DLIB.H)*.

While the program is running, you should monitor the counters with the standard visualization tools of the CS7DLIB. Since updating in this case takes place only every 100 ms, the macroscopic behaviour can be monitored and interpreted; it is best to analyse this behaviour using the debugger.

Information about CS7DLIB test system

- Timers are only ever updated between cycles.
- The jump to the time levels always takes place between cycles (no interruption of cycle)
- The transfer between I/O and process image must be parameterized, if desired, by means of user_cfg.c.

3.3 Generating and loading a C block

3.3.1 Generating a C block

Compiling and linking program modules

To run your application on the PLC, you must first compile your program modules, link them to the required runtime libraries and convert the linker file generated in this way into a loadable format.

The *abmain.bat* control file provided for this purpose can be found in directory *..\project*. This file controls compilation, linking and locating by means of the tasking tool chain.

References: /BSO/, Users Guide

Compiling and linking C blocks

The block generator *ab_gen.exe* uses this to generate a loadable C program block which is transferred to the PLC by means of loading tool *downplc.exe*.

Generating symbol file

In addition, the symbol preprocessor *sp166ta.exe* generates a symbol file for the on-line monitor after locating (see Section 3.4 On-line monitor), so that it is possible to access memory addresses symbolically.

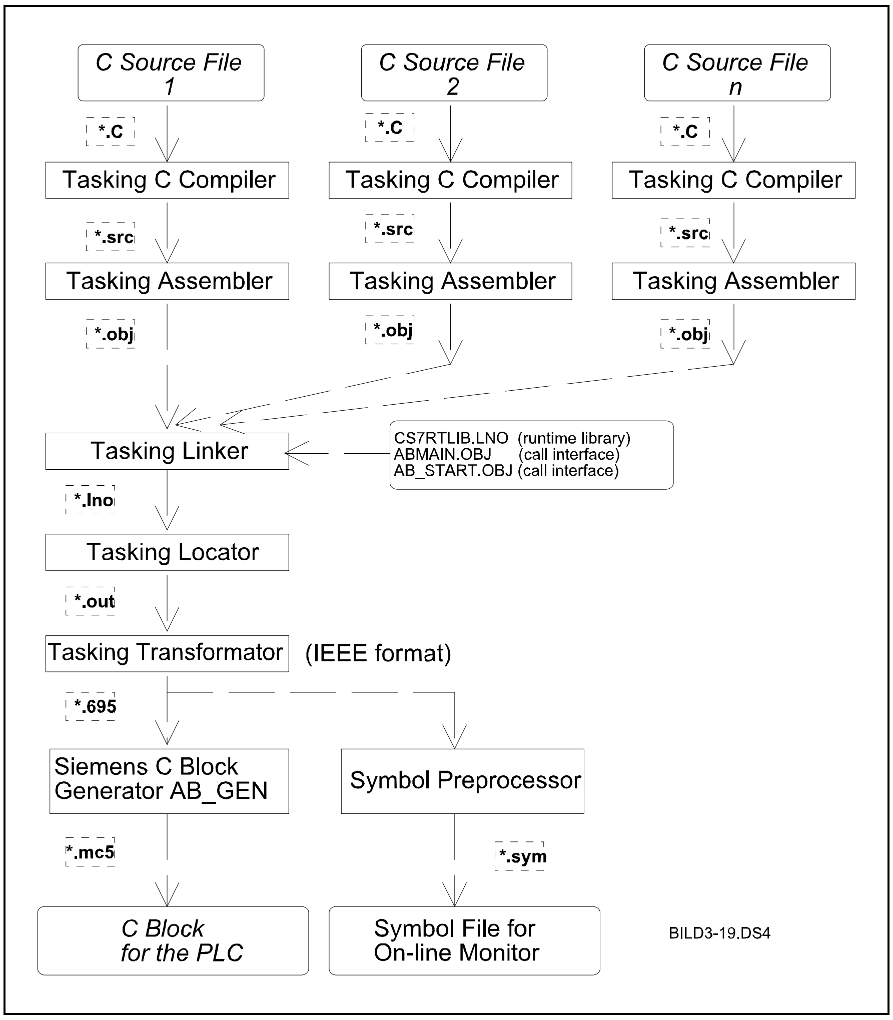


Fig. 3-19 Generating sequence for C program block and symbol file

Control file *abmain.bat* serves as the basis for your application. If the C application includes other modules, these only need to be added to the compiler and linker sections.

The return to the project directory must be added to the end of control file *abmain.bat*.

Note

In the event of warnings or error messages, please see Section 5.2
Response to errors or

References: /BSO/, Users Guide

3.3.2 Loading a C block

Memory allocation on the PLC

The memory of the PLC can be enabled explicitly via machine data. Changes to these machine data only become effective after general resetting of the PLC. The enabling of C memory reduces the available STEP 7 memory. If the machine data settings cause overlapping of the memory areas, the C memory has a higher priority than the STEP 7 memory.

STEP 7 memory

The size of the **STEP 7 memory** can be set in the general machine data $ON_PLC_USER_MEM_SIZE.

The default setting is 2, i.e. 64 Kbytes of STEP 7 memory are enabled.

```
AS314:  $ON_PLC_USER_MEM_SIZE       0 to 3    [0 to 3*220 KB]
AS315:  $ON_PLC_USER_MEM_SIZE       0 to 6    [0 to 6*220 KB]
```

Owing to the special memory allocation method of STEP 7, the memory requirements is as follows depending on the machine data:

Table 3-7 Memory requirements of the STEP 7 memory

$ON_PLC_USER_MEM_SIZE	Memory requirements
0	0 Kbytes
1	220 Kbytes
2	440 Kbytes
3	660 Kbytes
4	880 Kbytes
5	1100 Kbytes
6	1320 Kbytes

C memory

The size of the **C memory** can be set in the general machine data $ON_PLC_C_USER_MEM_SIZE.

The default setting is 0, i.e. no C memory is enabled.

```
AS314:  $ON_PLC_C_USER_MEM_SIZE     0 to 7    [0 to 7*64 KB]
AS315:  $ON_PLC_C_USER_MEM_SIZE     0 to 14   [0 to 14*64 KB]
```

Example for PLC memory allocation

Example for PLC memory allocation:

```
$ON_PLC_USER_MEM_SIZE       = 1
$ON_PLC_C_USER_MEM_SIZE     = 4
```

This results in the following PLC memory allocation:

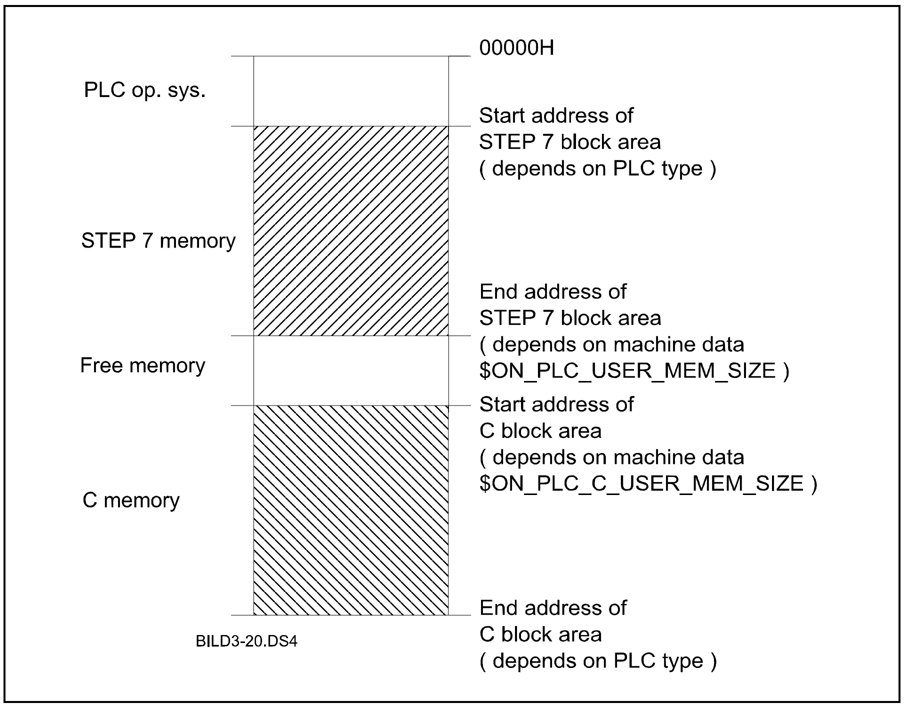


Fig. 3-20 PLC memory allocation

Locating the C block

The start address of the C block area is read from communication data block DB70 with the tool *rddbbplc.exe*. On this basis, the code and data segments of the application are assigned to the appropriate memory areas.

The memory areas and data pages are defined automatically by the *bs_addr.exe* tool, according to the PLC type and the area sizes specified for the code and data segments when control file *abmain.bat* is called. The data and code segments for the application are allocated to different memory areas, depending on whether or not an alternating buffer is used to load the block on the PLC.

Locating with alternating buffer activated

When the alternating buffer is active, the code section is switched over alternately. The data section is permanently located at the end of the C block area. (see Fig. 3-21 Allocation of C memory with active alternating buffer).

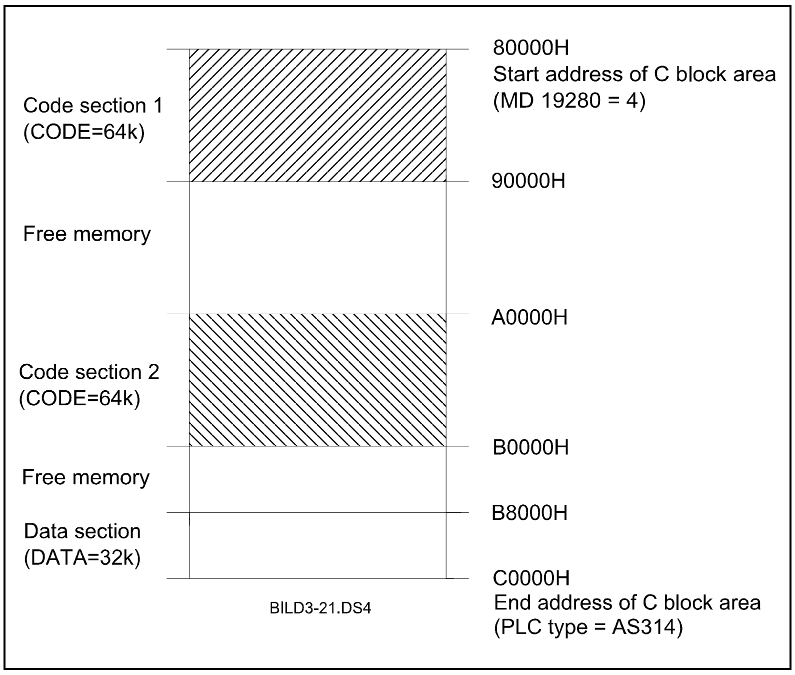


Fig. 3-21 Allocation of C memory with active alternating buffer using the example of an AS314 (also applies to AS315)

Locating with alternating buffer deactivated

When the alternating buffer is deactivated, the code section always begins at the start address of the C block area. The data section is located directly after the code section. If a monitor block is installed, it is located permanently at the end of the C block area (see Fig. 3-22 Allocation of C memory).

!

Important

The alternating buffer must always be deactivated when the monitor block is loaded.

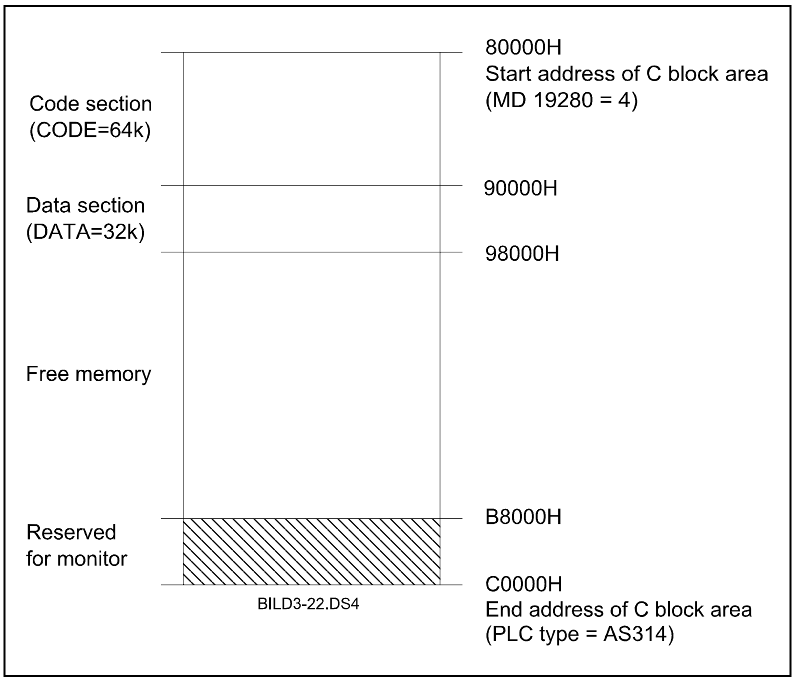


Fig. 3-22 Allocation of C memory with deactivated alternating buffer using the example of an AS314 (also applies to AS315)

Loading the C program

The C block *"abmain.mc5"* generated by block generator *ab_gen.exe* is divided into blocks (*"ab00.1vx")* of 64 Kbyte each by the generator. These blocks are loaded successively onto the PLC by means of tool *downplc.exe*, and are linked into the PLC when the last block has been loaded.

The C program is loaded onto the PLC automatically at the end of control file *abmain.bat*. It can be loaded separately by calling control file *asmload.bat* in directory *CS7TOOLS\MPI*.

Reloading of the C program after changes to the data declaration may only take place when the PLC is in the STOP state. The C program on the PLC can only be erased through a PLC overall memory reset.

Loading the PLC basic program

The PLC basic program is included in the scope of delivery (*CS7TOOLS\GP840D* directory). The above directory is set up for you automatically when the development environment is installed on your computer. This directory also includes control file *awlload.bat*. When this file is called, the compiled blocks of the PLC basic program are loaded on the PLC.

The C program can be executed on the PLC only if the PLC basic program is loaded. The C program is called by the PLC basic program. If the C program cannot be processed properly or the C program block is not installed, then the PLC switches to the STOP operating state (check entry in MW 40 to 52, see Section 5.2 Response to errors).

3.4 On-line monitor

The on-line monitor can be used to test the C block on the PLC.

HITEX monitor

This monitor is based on the remote debugger "telemon 80C166/167 (telemon 80C167)" with dialog software HiTOP supplied by HITEX (licence HITEX).

References: /HITEX/, User Manual

The dialog software is mouse-oriented with pull-down menus and flexible window system. It is installed on a DOS-PC (>=386) with MPI interface.

The tool SP166TA.EXE for editing the C166 Locator Formats for debug operation is supplied with the dialog software (see Section 3.3 Generating and loading a C block). For further details about installation and operator control, please refer to

References: /HITEX/, User Manual

The monitor is a remote debugger. The full range of functions for a remote debugger is divided between two programs, i.e. the monitor block and the user interface. The two programs operate on separate processors which are interlinked via a defined interface. The monitor and user interface communicate via debug data block DB71.

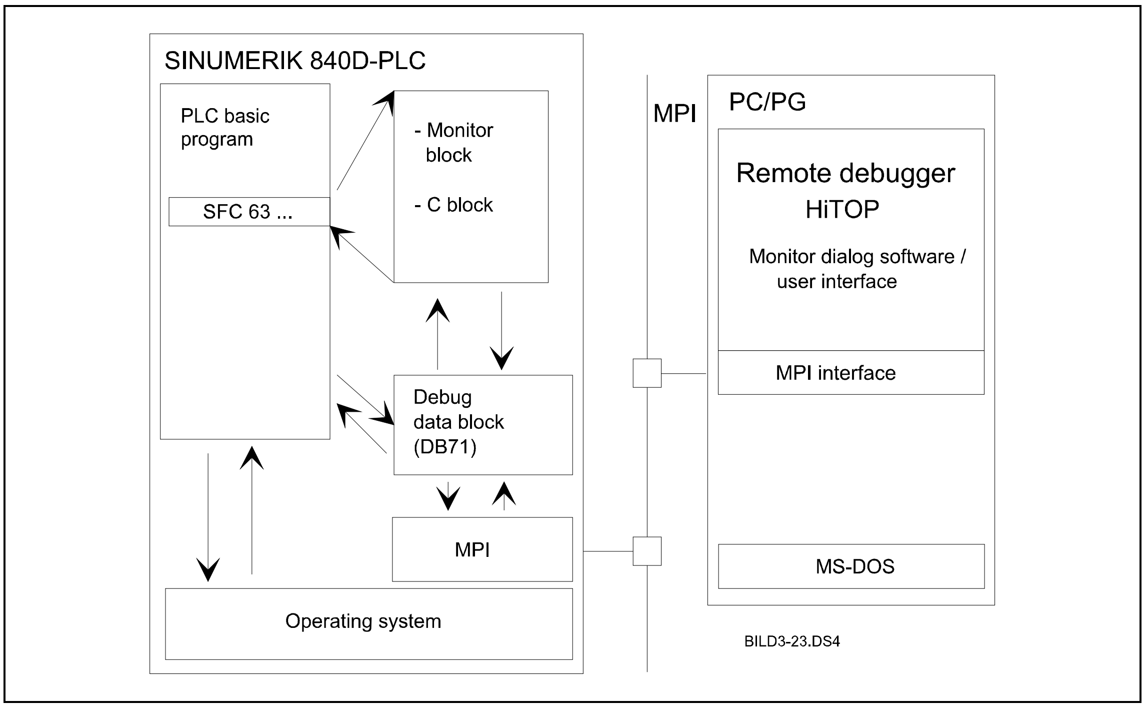


Fig. 3-23 On-line monitor

Loading the monitor block

In order to activate the monitor, you must first load the monitor block into the PLC. This is performed by means of control file *monload.bat.*

Note

The monitor block can be loaded only if alternating buffer operation is deselected.

Activating the monitor

The monitor is activated on the basis of data bit 500.0 in communication data block DB 70:

DB70 DBX500.0	= 0	Normal operating mode
	= 1	Debug mode with HITEX monitor

Note

After setting this data bit, it is absolutely essential to execute a complete PLC restart because the monitor block is activated in OB100.

Starting the monitor

The HiTOP dialog software is started with control file *monstart.bat*. This file activates the MPI interface and calls the dialog software *hit_167.exe*. When the dialog software is started, the HiScript file *startup.scr* is called. This presets certain variables for the interface and loads the symbol table. You can use files *startup.scr* and *hit_167.cfg* to customize the dialog software.

After successful initialization of the monitor block (in event of an error, note MW 50 and MW 52), cyclical operation is initiated.

Note

The C block is not operative after initialization of the monitor block.

The C block can be controlled after activation of the monitor block and start of the dialog interface.

Caution

When testing C blocks with the on-line monitor, please note the following points:

- If you set a breakpoint when the machine is in operation, the C block will no longer be processed cyclically (if, for example, a motional function was started before a breakpoint, then this can neither be checked nor stopped.).
  For this reason, the PLC basic program is stopped and the command output disabled when a breakpoint (except for start breakpoint) is reached.
  However, cyclic processing of the monitor block continues, meaning that the C block can be checked again. This state remains valid until the PLC is next reset.
- You can change memory cells in the RAM selectively using the monitor. The monitor does not check the location at which it writes the values. If you change the system data, this will cause the system to crash.

Breakpoint handling

There are 3 different types of breakpoint:

- User breakpoint; this is set by the user

- Temporary breakpoint; this is set by the monitor in single step mode
- Start breakpoint; this is set as soon as you call the user interface on the computer

If the C block to be tested encounters a breakpoint, program processing is halted at this point. Processing of the C block continues as soon as a RUN or STEP command is entered via the user interface.

Deactivating the monitor

To deactivate the monitor and reload a C block:

- Set DB70.DB500.0 = 0
- Load the C block
- execute a PLC overall memory reset

Notes on monitor operation

- System times and STEP 7 runtime levels remain active even if the C application has reached a stop point.
- Prior to every debug session, the C block must be reloaded on the PLC and a PLC reset executed.
- The on-line monitor supports debugging only in the basic cycle (= StdApplCycle()). The C alarm runtime levels are not processed in monitor operation.

User-related functions

The most important functions are given below:

Table 3-8 User-related functions

Main menu	**Submenu**	**Function**	**Key combination**
File	Load	Load symbol file for View List window File name: Symbolfile.sym File type: SYMBOLS All other input fields/options are irrelevant	FL
	Info	Status information	FI
	DOS Shell	Call DOS environment	FD
	Quit	End	FQ
Define	Breakpoint	Define stop point	DB
View	List	Display source text display of C program	VL
	Instruction	Display and modify code in assembler representation	VI
	Memory	Display RAM	VM
	Watch	Display and modify monitoring points	VW
Go	Next	Execute a machine command	F6
	Into	Command execution up to next C line, this command can also be used to branch to functions	F7
	Out of ...	This command is used to exit a function which you have "entered" with Go Into.	Shift F7
	Line	Command execution up to next C line, function calls are skipped	F8
	Run	Start program execution	F9
	Halt	Stop program execution	Shift F9
	Until...	Program execution up to a certain address	GU
Options	Update	Setting to define which window must be 'refreshed' cyclically.	OU
	Symbols	Display symbols ON/OFF	OY
	Screen	Change text resolution	OS

It is possible to call a local submenu within each window using the <spacebar> or the right-hand mouse button.

Within most input fields, it is possible to open an additional selection menu (e.g. wildcard sample for breakpoints, watches, etc.) using the <+> key or the left-hand mouse key.

When you exit from the operator interface <FQ>, you can save the current screen settings in a restore file. In order to restore these settings again at the next session, you must modify file *hit_167.cfg* (switch *-r*).

For a detailed description of the HiTOP dialog software, please refer to

References: /HITEX/, User Manual

C Call Interface for the Basic PLC Program 4

4.1 General information

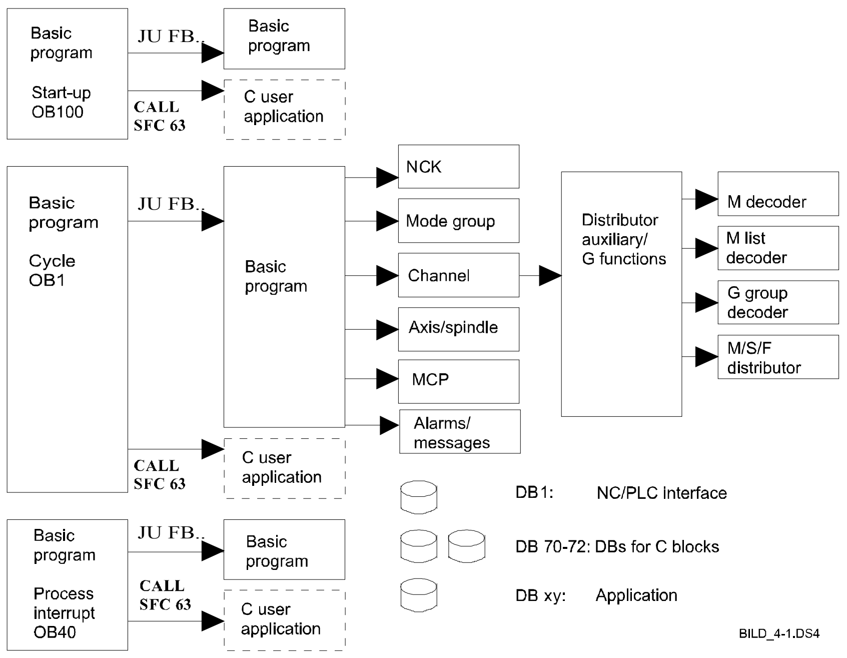


Fig. 4-1 Structure of 840D standard basic program (for details about function blocks shown with unbroken lines, refer to **References:** /PLCGP/, Description of Functions: Standard Machine). C functions for accessing NCK functionality are available for the C extensions (blocks with broken lines).

The C program file gp840d.c (header: gp840d.h) provides the C programmer with access to NCK functionality. The functions from gp840d.c represent a call interface of the basic PLC program. This call interface supplies the corresponding basic program block with transfer parameters, calls the block and transfers the return values back to the C function, i.e. the basic program block is always processed.

In chronological terms, the C function is processed before the basic program block (and therefore also receives/transmits values first). Owing to this mechanism, the return values of the C functions do not become valid until one cycle has elapsed after initiation of the function. To ensure, however, that the C function does not return any invalid values, all return values are set to 0 when a function is activated (positive edge at start input).

Note

A basic program block may only be called once per cycle if the C interface is in use.

For a description of the basic PLC program, please refer to

References: /PLCGP/, Description of Functions: Standard Machine

4.2 Description of the C functions

4.2.1 RUN_UP, start-up function

Function description

During start-up, the NCK and the PLC are synchronized, and the data blocks for the NCK/PLC application interface are generated in accordance with the NCK configuration stored in the machine data.
The RUN_UP function calls up the main function block FB1. For a description of FB1 please refer to

References: /PLCGP/, Description of Functions Standard Machine

Parameters

Table 4-1 Parameters for RUN_UP

Signal	Type	Value range	Remarks
MCPNum	WORD	0 to 2	No. of active MCPs 0: no MCP exists.
MCP1In MCP2In	S7_POINTER	I0.0 to I120.0 or F0.0 to F248.0	Start address for the input signals of the machine control panel
MCP1Out MCP2Out	S7_POINTER	Q0.0 to Q120.0 or F0.0 to F248.0	Start address for the output signals of the machine control panel
MCP1StatSend MCP2StatSend	S7_POINTER	Q0.0 to Q124.0, F0.0 to F252.0 or DBn.DBX0.0 to DBXm.0	Start address for the status doubleword for sending to the machine control panel: DW#16#08000000: time-out expired, else 0
MCP1StatRec MCP2StatRec	S7_POINTER	Q0.0 to Q124.0, F0.0 to F252.0 or DBn.DBX0.0 to DBXm.0	Start address for the status doubleword for receiving from the machine control panel: DW#16#00040000: time-out expired, else 0
MCP1BusAdr MCP2BusAdr	WORD		Bus address of the machine control panel
MCP1Timeout MCP2Timeout	ULONG	Recommended: 700 ms	Cyclical sign-of-life monitoring for the machine control panel
MCP1Cycl MCP2Cycl	ULONG	Recommended: 200 ms	Time frame for cyclical update of signals to machine control panel
BHG	WORD		Handheld operator panel interface: 0 - no handheld OP 1 - handheld OP on MPI 2 - handheld OP on MCP
BHGIn	S7_POINTER		Start address for data received on the PLC from the handheld operator panel
BHGOut	S7_POINTER		Start address for data sent from the PLC to the handheld operator panel
BHGStatSend	S7_POINTER	Q0.0 to Q124.0, F0.0 to F252.0 or DBn.DBX0.0 to DBXm.0	Start address for the status doubleword for sending to the handheld operator panel: DW#16#08000000: time-out expired, else 0

Signal	Type	Value range	Remarks
BHGStatRec	S7_POINTER	Q0.0 to Q124.0, F0.0 to F252.0 or DBn.DBX0.0 to DBXm.0	Start address for the status doubleword for receiving from the handheld operator panel: DW#16#00040000: time-out expired, else 0
BHGInLen	BYTE	Handheld OP default: B#16#6 (6 Byte)	No. of data items received from the handheld operator panel
BHGOutLen	BYTE	Handheld OP default: B#16#14 (20 Byte)	No. of data items sent to the handheld operator panel
BHGTimeout	ULONG	Recommended: 700 ms	Cyclical sign-of-life monitoring for the handheld operator panel
BHGCycl	ULONG	Recommended: 400 ms	Time frame for cyclical update of signals to handheld operator panel
BHGRecGDNo	WORD	Hheld OP default: 2	Receive GD circle no.
BHGRecGBZNo	WORD	Hheld OP default: 1	Receive GBZ no.
BHGRecObjNo	WORD	Hheld OP default: 1	Object no. for receive GBZ
BHGSendGDNo	WORD	Hheld OP default: 2	Send GD circle no.
BHGSendGBZNo	WORD	Hheld OP default: 2	Send GBZ no.
BHGSendObjNo	WORD	Hheld OP default: 1	Objekt no. for send GBZ
NCCyclTimeout	ULONG	Recommended: 200 ms	Cyclical sign-of-life monitoring for NCK
NCRunupTimeout	ULONG	Recommended: 50 s	Start-up monitoring for NCK
ListMDecGrp	WORD	0..16	Activation of extended M group decoding. 0 = not active 1..16: no. of M groups
NCKomm	VKE_TYPE		PLC-NC communication services (FB 2/3/4: Put/Get/PI) true: active
MMCToIF	VKE_TYPE		Transfer of MMC signals to the interface (modes, program modification, etc.) true : active
HWheelMMC	VKE_TYPE		True: Handwheel selection via MMC. False: handwheel selection via user program
MsgUser	WORD	0..25	No. of user areas for messages (DB 2)
UserIR	VKE_TYPE		OB40 local data extension required for processing signals from user there
IRAuxfuT	VKE_TYPE		Evaluate T function on OB40
IRAuxfuH	VKE_TYPE		Evaluate H function on OB40
IRAuxfuE	VKE_TYPE		Evaluate E function on OB40

Programming example

```
/* Programming example for RUN_UP */

VOID Bsp_RUN_UP( VOID )
{
S7_POINTER   MCP1In, MCP1Out, MCP1StatSend, MCP1StatRec,
             unused;

MCP1In.memArea             = INPUT;
MCP1In.byteNo       = 0;
MCP1Out.memArea            = OUTPUT;
MCP1Out.byteNo             = 0;
MCP1StatSend.memArea       = OUTPUT;
MCP1StatSend.byteNo        = 8;
MCP1StatRec.memArea        = OUTPUT;
MCP1StatRec.byteNo         = 12;
unused.memArea             = 0;
unused.byteNo       = 0;

RUN_UP(      1,
             MCP1In, MCP1Out, MCP1StatSend, MCP1StatRec,
             6, 700, 200,
             unused, unused, unused, unused,
             0, 700, 200,
             0,
             unused, unused, unused, unused,
             20, 6, 700, 400, 2, 1, 1, 2, 2, 1,
             200, 50000UL,
             1,
             VKE_TRUE, VKE_TRUE, VKE_TRUE,
             10,
             VKE_FALSE, VKE_FALSE, VKE_FALSE, VKE_FALSE );
}
```

4.2.2 GET, read NCK variables

Function description

The GET function can be used to read variables from the NCK area. The variables addressed by Addr[8] are copied to the referenced data block following a successful read operation.

To reference the variables, all the required variables are first selected with the NCK VAR selector tool, and then generated as an STL source in a data block. In the C user program, structures of the NCK_VAR type are now filled with the generated values.

For some variables, it is necessary to select the unit and/or the line or column. It is possible to select a base type for these variables; i.e. unit/column/line are initialized with “0”.

The value for this is taken from input parameters Unit[8]/Column[8]/Line[8]. The GET function calls basic function block FB2. For a description of FB2 and the NCK VAR selector, please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-2 Parameters for GET

Signal	Type	Value range	Remarks
Req	VKE_TYPE		Start job on positive edge
NumVar	UWORD	1..8	No. of variables to be read
Addr[8]	NCK_VAR		Variable names from NCK-VAR selector
Unit[8]	UBYTE		Unit address, optional for variable addressing
Column[8]	UWORD		Column address, optional for variable addressing
Line[8]	UWORD		Line address, optional for variable addressing
RD[8]	S7_ANY_POINTER	P#DBnr.dbxm.n	Line area for read data

Return parameters

Table 4-3 Return parameters for GET

Signal	Type	Value range	Remarks
Error	VKE_TYPE		Job was given negative acknowledgement or could not be executed
NDR	VKE_TYPE		Job was successfully executed. Data are available
State	UWORD		See error codes

Error codes

If a job could not be executed, this is indicated by a ‘1’ in the state parameter. The cause of the error is coded in the State block output:

Table 4-4 Error codes for GET

State		Meaning	Information
WORD-H	WORD-L		
1 to 8	1	Access error	In high-byte number of variable in which error occurred
0	2	Error in job request	Incorrect variable syntax in job
0	3	Negative acknowledgement, job not executable	Internal error, possible remedy: Reset NC
1 to 8	4	Insufficient local user memory available	Variable read is longer than definition in RD[8]; in high-byte number of variable in which error occurred.
0	5	Format conversion error	Error on conversion of double variable type: Var. not within range of S7-REAL
0	6	FIFO full	Job must be repeated, because queue is full
0	7	Option not enabled	GP parameter “NCKomm” is not enabled
1 to 8	8	Incorrect dest. area (RD)	RD[8] cannot be local data
0	9	Communication busy	Job must be repeated
1 to 8	10	Error on variable addressing	Unit or Column/Line contain value 0
0	11	Variable address invalid	Check Addr[8]

Programming example

```
/* Programming example for GET */

VOID Bsp_GET( VOID )
{
GET_STAT            get_ret;
UWORD               i;
NCK_VAR             var[8];
UBYTE               unit[8];
UWORD               column[8];
UWORD               line[8];
S7_ANY_POINTER      dest[8];

for ( i=0; i < 8; i++ )
{
      /* R10 to R17 ( permanent addressing ) */
      var[i].syntax_id          = 0x82;
      var[i].bereich_u_einheit  = 0x41;
      var[i].spalte       = 0x1;
      var[i].zeile              = 10 + i+1;
      var[i].bausteintyp        = 0x15;
      var[i].zeilenanzahl       = 0x1;
      var[i].typ                = 0xf;
      var[i].laenge       = 0x8;
      unit[i]                   = 0;
      column[i]                 = 0;
      line[i]                   = 0;

      /* Destination for R10 to R17 */
      dest[i].type = TYP_BYTE;
      dest[i].count= var[i].laenge;
```

```
        dest[i].dbNo = DB_TEST;
        dest[i].offset= DB_TEST_OFS_GET + i*dest[i].count;
}

get_ret = GET(      E_R( 77, 7 ),/* Req */
                    8,           /* NumVar */
                    var,         /* Addr[] */
                    unit,        /* Unit[] */
                    column,      /* Column[] */
                    line,        /* Line[] */
                    dest   );    /* RD[] */

A_W( 112, 0, get_ret.Error );
A_W( 112, 1, get_ret.NDR );
MW_W( 100, get_ret.State );
}
```

Data block generated by NCK VAR selector with structure for R10:

```
STRUCT
rpa_10C1RP:
        STRUCT
        SYNTAX_ID : BYTE  := B#16#82;
        bereich_u_einheit : byte := B#16#41;
        spalte : word := W#16#1;
        zeile : word := W#16#11;
        bausteintyp : byte := B#16#15;
        ZEILENANZAHL : BYTE := B#16#1;
        typ : byte := B#16#F;
        laenge : byte := B#16#8;
END_STRUCT ;
```

4.2.3 PUT, write NCK variables

Description

The PUT function can be used to write variables into the NCK area. The variables addressed by Addr[8] are overwritten with the data in the data block referenced by SD[8].

To reference the variables, all the required variables are first selected with the NCK VAR selector tool, and then generated as an STL source in a data block. In the C user program, structures of the NCK_VAR type are now filled with the generated values.

For some variables, it is necessary to select the unit and/or the line or column. It is possible to select a base type for these variables; i.e. unit/column/line are initialized with “0”.

The value for this is taken from input parameters Unit[8]/Column[8]/Line[8]. The PUT function calls basic function block FB3. For a description of FB3 and the NCK-VAR selector, please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-5 Parameters for PUT

Signal	Type	Value range	Remarks
Req	VKE_TYPE		Start job on positive edge
NumVar	UWORD	1..8	No. of variables to be written
Addr[8]	NCK_VAR		Variable names from NCK VAR selector
Unit[8]	UBYTE		Unit address, optional for variable addressing
Column[8]	UWORD		Column address, optional for variable addressing
Line[8]	UWORD		Line address, optional for variable addressing
SD[8]	S7_ANY_POINTER	P#DBnr.dbxm.n	Data to be written

Return parameters

Table 4-6 Return parameters for PUT

Signal	Type	Value range	Remarks
Error	VKE_TYPE		Job was given negative acknowledgement or could not be executed
Done	VKE_TYPE		Job was successfully executed
State	UWORD		See error codes

Error codes

If a job could not be executed, this is indicated by a ‘1’ in the state parameter. The cause of the error is coded in the State block output:

Table 4-7 Error codes for PUT

State		Meaning	Information
WORD-H	WORD-L		
1 to 8	1	Access error	In high-byte number of variable in which error occurred
0	2	Error in job request	Incorrect variable syntax in job
0	3	Negative acknowledgement, job not executable	Internal error, possible remedy: Reset NC
1 to 8	4	Data areas or data types do not match	Check data to be written in SD[8]; in high-byte number of variable in which error occurred
0	6	FIFO full	Job must be repeated, because queue is full
0	7	Option not enabled	GP parameter “NCKomm” is not enabled
1 to 8	8	Incorrect dest. area (SD)	SD[8] cannot be local data
0	9	Communication busy	Job must be repeated
1 to 8	10	Error on variable addressing	Unit or Column/Line contain value 0
0	11	Variable address invalid	Check Addr[8]

Programming example

```
/* Programming example for PUT */

VOID Bsp_PUT( VOID )
{
PUT_STAT            put_ret;
UWORD               i;
NCK_VAR             var[8];
UBYTE               unit[8];
UWORD               column[8];
UWORD               line[8];
S7_ANY_POINTER      src[8];
S7_DB_HANDLE        dbtest = AUF_DB( DB_TEST );

for ( i=0; i < 8; i++ )
{
      /* R10 to R17 ( variable addressing ) */
      var[i].syntax_id          = 0x82;
      var[i].bereich_u_einheit  = 0;
      var[i].spalte      = 0;
      var[i].zeile              = 0;
      var[i].bausteintyp        = 0x15;
      var[i].zeilenanzahl       = 0x1;
      var[i].typ                = 0xf;
      var[i].laenge      = 0x8;
      unit[i]                   = 0x41;
      column[i]                 = 0x1;
      line[i]                   = 10 + i+1;

      /* Source for R10 to R17 */
      src[i].type  = TYP_BYTE;
      src[i].count = var[i].laenge;
      src[i].dbNo   = DB_TEST;
      src[i].offset= DB_TEST_OFS_PUT + i*src[i].count;

      /* Overwrite source with value */
      DD_W( dbtest, src[i].offset, F2L( 1.0 ) );
}

put_ret = PUT(      E_R( 77, 6 ),/* Req */
                    8,           /* NumVar */
                    var,         /* Addr[] */
                    unit,        /* Unit[] */
                    volumn,      /* Column[]*/
                    line,        /* Line[] */
                    src );       /* RD[] */

A_W( 112, 2, put_ret.Error );
A_W( 112, 3, put_ret.Done );
MW_W( 102, put_ret.State );
}
```

Data block generated by NCK VAR selector with structure for R10:

```
STRUCT
 rpa_10C1RP:
   STRUCT
   SYNTAX_ID : BYTE := B#16#82;
   bereich_u_einheit : byte := B#16#41;
   spalte : word := W#16#1;
   zeile : word := W#16#11;
   bausteintyp : byte := B#16#15;
   ZEILENANZAHL : BYTE := B#16#1;
   typ : byte := B#16#F;
   laenge : byte := B#16#8;
   END_STRUCT ;
```

Miscellaneous function for converting float to long:

```
VOID   F2L( FLOAT value )
{
P_USHORT     pointer;

      /* Cast pointer and generate return value */
      pointer = (USHORT *)&value;
      return ( (*pointer) * 0x10000UL + *(pointer+1) );
}
```

4.2.4 PI, general PI services

Description

The PI function can be used to start program instance services in the NCK area. The specified service is referenced in the PIService parameter (see gp840d.h for the defines). The selected PI service is supplied with parameters by means of the additional freely assignable input variables with different data types (Addr[4] for strings, WVar[6] for integers or word variables).

The PI function calls up basic function block FB4. For a description of FB4 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-8 Parameters for PI

Signal	Type	Value range	Remarks
Req	VKE_TYPE		Start job on positive edge
PIService	UWORD	1..16	PI service
Unit	UWORD	1..	Unit number
Addr[4]	S7_ANY_POINTER	P#DBnr.dbxm.n	Reference to string specification according to PI service selected
WVar[9]	WORD	1..	Word specification according to PI service selected

Return parameters

Table 4-9 Return parameters for PI

Signal	Type	Value range	Remarks
Error	VKE_TYPE		Job was given negative acknowledgement or could not be executed
Done	VKE_TYPE		Job was successfully executed
State	UWORD		See error codes

Error codes

If a job could not be executed, this is indicated by a ‘1’ in the state parameter. The cause of the error is coded in the State block output:

Table 4-10 Error codes for PI

State	Meaning	Information
3	Negative acknowledgement, job not executable	Internal error, possible remedy: Reset NC
6	FIFO full	Job must be repeated, because queue is full
7	Option not enabled	GP parameter “NCKomm” is not enabled
9	Communication busy	Job must be repeated

Programming example

```
/* Programming example for PI */

VOID Bsp_PI( VOID )
{
PI_STAT             pi_ret;
S7_ANY_POINTER      adr[4];
WORD                var[9];
UWORD               i;
S7_DB_HANDLE        dbtest       = AUF_DB( DB_TEST );
UWORD               ebtest       = EB_R( 72 );
BYTE   mpfpath[MAX_STR_LEN+1]    = "/_N_MPF_DIR/";
BYTE   mpfprog[MAX_STR_LEN+1]    = "_N_ASUP_MPF";

switch ( ebtest )
{
      /* PI_SELECT */
      case 1:
            /* Address for mpfpath */
            adr[0].type  = TYP_BYTE;
            adr[0].count = MAX_STR_LEN+2;
            adr[0].dbNo  = DB_TEST;
            adr[0].offset= DB_TEST_OFS_PI;

            /* Address for mpfprog */
            adr[1].type  = TYP_BYTE;
            adr[1].count = MAX_STR_LEN+2;
            adr[1].dbNo  = DB_TEST;
            adr[1].offset= DB_TEST_OFS_PI +adr[0].count;

            /* Write string for mpfpath */
            DS_W( dbtest, adr[0].offset, mpfpath );

            /* Write string for mpfpath */
            DS_W( dbtest, adr[1].offset, mpfprog );

            pi_ret = PI( E_R( 77, 1 ),/* Req */
                         PI_SELECT,   /* PIService */
                         1,           /* Unit */
                         adr,         /* Addr[] */
                         var    );    /* WVar[] */

            A_W( 113, 0, pi_ret.Error );
            A_W( 113, 1, pi_ret.Done );
            MW_W( 104, pi_ret.State );
            break;

      /* PI_CONFIG */
      case 2:
            var[0] = 1;  /* Classification */
```

```
              pi_ret = PI(  E_R( 77, 1 ),/* Req */
                            PI_CONFIG,    /* PIService */
                            1,            /* Unit */
                            adr,          /* Addr[] */
                            var   );      /* WVar[] */

              A_W( 113, 0, pi_ret.Error );
              A_W( 113, 1, pi_ret.Done );
              MW_W( 104, pi_ret.State );
              break;

       default:
              break;
}
}
```

Miscellaneous function for writing a string to a data block:

```
VOID   DS_W(  S7_DB_HANDLE db,
              USHORT       byteOffset,
              BYTE         value[]      )
{
UBYTE  i, str_len;

       /* Write string identifier to db */
       DB_W( db, byteOffset, 0x0E );
       /* Write string length to db */
       if ( ( str_len = strlen( value ) ) > MAX_STR_LEN )
              str_len = MAX_STR_LEN;
       DB_W( db, byteOffset+1, str_len );
       /* Write user data to db */
       for ( i=0; i < str_len; i++ )
              DB_W( db, i + byteOffset+2, value[i] );
}
```

4.2.5 GETGUD, read GUD variable

Function description

The GETGUD function can be used to read a GUD variable (GUD = **G**lobal **U**ser **D**ata) from the NCK or channel area. The GETGUD function calls up basic function block FB5. For a description of FB5 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-11 Parameters for GETGUD

Signal	Type	Value range	Remarks
Req	VKE_TYPE		Start job on positive edge
Addr[32]	BYTE		GUD variable name
Area	BYTE		Unit address: 0: NCK variable 2: Channel variable
Unit	BYTE		Unit NCK: Unit:=1 Unit channel: channel no.
Index1	WORD		Array index 1 of variable The value of the variable is 0 if the array index is not used
Index2	WORD		Array index 2 of variable The value of the variable ist 0 if the array index is not used
CnvtToken	VKE_TYPE		Activation of the generation of a variable token
RD	S7_ANY_POINTE R	P#DBnr.dbxm.n	Data to be read

Return parameters

Table 4-12 Return parameters for GETGUD

Signal	Type	Value range	Remarks
VarToken	NCK_VAR		Address of a 10-byte token
Error	VKE_TYPE		Job was given negative acknowledgement or could not be executed
Done	VKE_TYPE		Job was successfully executed. Data are available
State	UWORD		See error codes

Error codes

If a job could not be executed, this is indicated by a ‘1’ in the state parameter. The cause of the error is coded in the State block output:

Table 4-13

State		Meaning	Information
WORD-H	**WORD-L**		
0	1	Access error	
0	2	Error in job request	Incorrect variable syntax in job
0	3	Negative acknowledgement, job not executable	Internal error, possible remedy: Reset NC
0	4	Insufficient local user memory available	Check data to be read in RD
0	6	FIFO full	Job must be repeated, because queue is full
0	7	Option not enabled	GP parameter “NCKomm” is not enabled
0	8	Incorrect dest. area (RD)	RD cannot be local data
0	9	Communication busy	Job must be repeated
0	10	Error on addressing	Unit contains value 0
0	11	Variable address invalid	Check Addr (or variable name), Area, Unit

Programming example

```
/* Programming example for GETGUD */

VOID Bsp_GETGUD( VOID )
{
GETGUD_STAT         getgud_ret;
PUT_STAT            put_ret;
BYTE                var[MAX_STR_LEN+1] = "GUD_VAR_NCK";
S7_ANY_POINTER      dest;
UBYTE               unit[8];
UWORD               column[8];
UWORD               line[8];
S7_ANY_POINTER      src[8];
S7_DB_HANDLE        dbtest = AUF_DB( DB_TEST );
static UBYTE        cycle;
VKE_TYPE            put_start;

/* Destination for GUD variable */
dest.type    = TYP_BYTE;
dest.count   = 4;
dest.dbNo    = DB_TEST;
dest.offset  = DB_TEST_OFS_GETGUD;

getgud_ret = GETGUD(E_R( 73, 4 ),/* Req */
                    var,              /* Addr */
                    0,                /* Area */
                    1,                /* Unit */
                    0,                /* Index1 */
                    0,                /* Index2 */
                    E_R( 73, 5 ),/* CnvtToken */
                    dest   );         /* RD */

A_W( 112, 0, getgud_ret.Error );
A_W( 112, 1, getgud_ret.Done );
```

```
MW_W( 100, getgud_ret.State );
}


if (E_R( 73, 5 )
{
       /* Address for source */
       src[0].type  = TYP_BYTE;
       src[0].count = 4;
       src[0].dbNo  = DB_TEST;
       src[0].offset= DB_TEST_OFS_GETGUD+4;

       /* Overwrite source with value */
       DD_W( dbtest, src[0].offset, 2UL );

       /* Supply values for variable addressing */
       unit[0]      = 0;
       column[0]    = 0;
       line[0]      = 0;

       /* 1 PLC cycle delay for put_start */
       if ( getgud_ret.Done )
              if ( cycle )
                     put_start = VKE_TRUE;
              else
                     cycle = 1;
       else
              cycle = 0;

       put_ret = PUT(put_start,         /* Req */
                     1,                 /* NumVar */
                     &getgud_ret.VarToken,/* Addr[] */
                     unit,              /* Unit[] */
                     volumn,            /* Column[]*/
                     line,              /* Line[] */
                     src );             /* RD[] */

       A_W( 112, 2, put_ret.Error );
       A_W( 112, 3, put_ret.Done );
       MW_W( 102, put_ret.State );
}
```

4.2.6 ASUP, start asynchronous subroutines

Function description

The ASUP function can be used to initiate any functions on the NC. In order to be started from the PLC, an ASUP must be selected and configured by an NC program. An ASUP which has been prepared in this way can be started at any time from the PLC. The ASUP calls up basic function block FC9. For a description of FC9 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-14 Parameters for ASUP

Signal	Type	Value range	Remarks
Start	VKE_TYPE		
ChanNo	UWORD	1, 2	No. of NC channel
IntNo	UWORD	1..8	Interrupt no.

Return parameters

Table 4-15 Return parameters for ASUP

Signal	Type	Value range	Remarks
Activ	VKE_TYPE		1 = active
Done	VKE_TYPE		1 = ASUP terminated
Error	VKE_TYPE		
StartErr	VKE_TYPE		1 = Interrupt number not assigned

Programming example

```
/* Programming example for ASUP */

VOID Bsp_ASUP
{
ASUP_STAT    asup_ret;

asup_ret = ASUP(    E_R( 77, 5 ), /* Start */
                    1,                  /* ChanNo */
                    EB_R( 72 )    );    /* IntNo */

A_W( 112, 7, asup_ret.Activ );
A_W( 112, 6, asup_ret.Done );
A_W( 112, 5, asup_ret.Error );
A_W( 112, 4, asup_ret.StartErr );
}
```

4.2.7 AL_MSG, error messages and operational messages

Function description

The AL_MSG function evaluates the signals entered in DB 2, and displays them as incoming and outgoing error messages and operational messages on the MMC. The AL_MSG function calls basic function block FC10. For a description of FC10 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-16 Parameters for AL_MSG

Signal	Type	Value range	Remarks
ToUserIF	VKE_TYPE		1 = Transfer of signals to application interface each cycle
Quit	VKE_TYPE		1 = Error message acknowledgement

Programming example

```
/* Programming example for AL_MSG */

VOID Bsp_AL_MSG
{
S7_DB_HANDLE  db2 = AUF_DB( 2 );

D_W( db2, 1, 0, E_R( 77, 3 ) );  /* BM 510008 */
D_W( db2, 0, 0, E_R( 77, 4 ) );  /* FM 510000 */

AL_MSG(      VKE_FALSE,          /* ToUserIF */
             E_R( 77, 2 ) );     /* Quit */
}
```

4.2.8 BHGDisp, display control for the handheld operator panel

Function description

This function controls the display of the handheld operator panel. The BHGDisp function calls up basic function block FC13. For a description of FC13 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-17 Parameters for BHGDisp

Signal	Type	Value range	Remarks
Row	UBYTE	0..3	Display row 0: no display output 1: Row 1 2: Row 2 3: Row 1 and row 2
ChrArrray[32]	BYTE	-	Complete display contents
Convert	VKE_TYPE		Activation of numeric conversion.
Addr	S7_POINTE R	-	Points to variable to be converted
DataType	UBYTE	1..8	Data type of variable 1: Bool, 1 character 2: Byte, 3 characters 3: Char, 1 character 4: Word, 5 characters 5: Int, 6 characters 6: Dword, 7 characters 7: Dint, 8 characters 8: Real, 9 characters (see Parameter Digits)
StringAddr	UWORD	1..32	Address within variable ChrArray
Digits	UBYTE	1..3	Only relevant for data type Real with leading sign (LS) 1 : 6.1 digits without LS 2 : 5.2 digits without LS 3 : 4.3 digits without LS

Return parameters

Table 4-18 Return parameters for BHGDisp

Signal	Type	Value range	Remarks
Error	VKE_TYPE		Conversion error, number overflow or StringAddr error

Programming example

```
/* Programming example for BHGDisp */

VOID Bsp_BHGDisp( VOID )
{
BHGDISP_STAT        bhgdisp_ret;
static UWORD        cycle;
S7_POINTER          var1, var2;
BYTE   bhgdisp[MAX_STR_LEN+1] = "Zeile 1---------Zeile 2---
------";
S7_DB_HANDLE        dbtest = AUF_DB( DB_TEST );

/* Assign address for var1 */
```

```
var1.memArea = DATABLOCK;
var1.byteNo  = DB_TEST_OFS_BHG+0;
var1.dbNo    = DB_TEST;

/* Assign address for var2 */
var2.memArea = DATABLOCK;
var2.byteNo  = DB_TEST_OFS_BHG+4;
var2.dbNo    = DB_TEST;

/* Write values for var1 and var2 to dbtest */
if ( E_R( 76, 5 ) )
{
       DD_W( dbtest, var1.byteNo, F2L( -0.5 ) );
       DD_W( dbtest, var2.byteNo, F2L( -10.0 ) );
}
else
{
       DD_W( dbtest, var1.byteNo, F2L( 0.5 ) );
       DD_W( dbtest, var2.byteNo, F2L( 10.0 ) );
}

switch ( cycle )
{
       case 1:
              /* Write first display row */
              bhgdisp_ret = BHGDisp(1,   /*Row*/
                            bhgdisp,     /* ChrArray */
                            E_R( 76, 4 ),/* Convert */
                            var1,        /* Addr */
                            8,           /* DataType */
                            16,          /* StringAddr */
                            3      );    /* Digits */

              A_W( 113, 6, bhgdisp_ret.Error );

              cycle = 2;
              break;

       case 2:
              /* Write second display row */
              bhgdisp_ret = BHGDisp(2,   /*Row*/
                            bhgdisp,     /* ChrArray */
                            E_R( 76, 4 ),/* Convert */
                            var2,        /* Addr */
                            8,           /* DataType */
                            32,          /* StringAddr */
                            3      );    /* Digits */

              A_W( 113, 6, bhgdisp_ret.Error );

              cycle = 0;
```

```
                break;

        default:
                cycle = 1;
                break;
}
}
```

Miscellaneous function for converting float to long:

```
VOID   F2L( FLOAT value )
{
P_USHORT     pointer;

        /* Cast pointer and generate return value */
        pointer = (USHORT *)&value;
        return ( (*pointer) * 0x10000UL + *(pointer+1) );
}
```

4.2.9 POS_AX, positioning of linear and rotary axes

Function description

The POS_AX function can be used to traverse NC axes, which have been defined in machine data as "concurrent axes", from the PLC. During normal operation, these axes can also be traversed using the JOG keys. The POS_AX function calls up basic program block FC15. For a description of FC15 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-19 Parameters for POS_AX

Signal	Type	Value range	Remarks
Start	VKE_TYPE		
AxisNo	UWORD	1..8	No. of axis to be traversed
IC	VKE_TYPE		0 = absolute 1 = incremental
Inch	VKE_TYPE		0 = mm 1 = inch
HWheelOv	VKE_TYPE		1 = handwheel overlay
Pos	FLOAT	± 0.1469368 E -38 to ± 0.1701412 E +39	Linear axis: mm Rotary axis: degrees
FRate	FLOAT	± 0.1469368 E -38 to ± 0.1701412 E +39	Linear axis: mm/min Rotary axis: degrees/min

Return parameters

Table 4-20 Return parameters for POS_AX

Signal	Type	Value range	Remarks
InPos	VKE_TYPE		1 = Position reached
Activ	VKE_TYPE		1 = active
StartErr	VKE_TYPE		Axis cannot be started
Error	VKE_TYPE		Error on traversal

Programming example

```
/* Programming example for POS_AX */

VOID Bsp_POS_AX( VOID )
{
POS_AX_STAT   pos_ax_ret;

pos_ax_ret = POS_AX(      E_R( 76, 7 ),/*Start*/
                          5,           /* AxisNo */
                          E_R( 76, 6 ),/* IC */
                          VKE_FALSE,   /* Inch */
                          VKE_FALSE,   /* HWheelOv */
                          -777.0,      /* Pos */
                          3333.0 );    /* FRate */
A_W( 104, 7, pos_ax_ret.InPos );
A_W( 104, 6, pos_ax_ret.Activ );
A_W( 104, 5, pos_ax_ret.StartErr );
A_W( 104, 4, pos_ax_ret.Error );
}
```

4.2.10 PART_AX, positioning of indexing axes

Function description

The PART_AX function can be used to traverse NC axes, which have been defined in machine data as "indexing axes", from the PLC. During normal operation, these axes can also be traversed using the JOG keys. The PART_AX function calls up basic program block FC16. For a description of FC16 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-21 Parameters for PART_AX

Signal	Type	Value range	Remarks
Start	VKE_TYPE		
AxisNo	UWORD	1..8	No. of axis to be traversed
IC	VKE_TYPE		0 = abs. direction 1 = incremental direction
DC	VKE_TYPE		0 = defined direction 1 = shortest path
Minus	VKE_TYPE		in preparation
Plus	VKE_TYPE		in preparation
Pos	WORD	0 to +32767	Indexing position no.
FRate	FLOAT	± 0.1469368 E -38 to ± 0.1701412 E +39	Linear axis: mm/min Rotary axis: degrees/min

Return parameters

Table 4-22 Return parameters for PART_AX

Signal	Type	Value range	Remarks
InPos	VKE_TYPE		1 = Position reached
Activ	VKE_TYPE		1 = active
StartErr	VKE_TYPE		Axis cannot be started
Error	VKE_TYPE		Error on traversal

Programming example

```
/* Programming example for PART_AX */
VOID Bsp_PART_AX( VOID )
{
PART_AX_STAT  part_ax_ret;
part_ax_ret = PART_AX(     E_R( 76, 3 ),/*Start*/
                           6,           /* AxisNo */
                           E_R( 76, 2 ),/* IC */
                           VKE_FALSE,   /* DC */
                           VKE_FALSE,   /* Minus */
                           VKE_FALSE,   /* Plus */
                           EB_R( 72 ),  /* Pos */
                           1111.0 );    /* FRate */
A_W( 105, 7, part_ax_ret.InPos );
A_W( 105, 6, part_ax_ret.Activ );
A_W( 105, 5, part_ax_ret.StartErr );
A_W( 105, 4, part_ax_ret.Error );
}
```

4.2.11 YDelta, star/delta selection

Function description

The function for star/delta selection is performed in both directions by defined (time-controlled) selection logic. The function can only be used for digital main spindle drives, and must be called up separately for each spindle. The YDelta function calls up basic function block FC17. For a description of FC17 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-23 Parameters for YDelta

Signal	Type	Value range	Remarks
YDelta	VKE_TYPE		0 = star 1 = delta The selection edge of the signal initiates the selection.
SpindleIFNo	UWORD	1..	Number of spindle interface
TimeVal	ULONG	0..	Selection time
TimerNo	UWORD	10..	Timer for programming the wait time.

Return parameters

Table 4-24 Return parameters for YDelta

Signal	Type	Value range	Remarks
Y	VKE_TYPE		Activation of the star contactor
Delta	VKE_TYPE		Activation of the delta contactor

Programming example

```
/* Programming example for Ydelta */

VOID Bsp_YDelta
{
YDELTA_STAT   ydelta_ret;
S7_DB_HANDLE  db34 = AUF_DB( 34 );

ydelta_ret = YDelta(       E_R( 77, 0 ),/*YDelta*/
                           4,           /* SpindleIFNo */
                           3000,        /* TimeVal (3s) */
                           10    );     /* TimerNo */

D_W( db34, 21, 3, VKE_FALSE );
/* Acknowledge motor selection */

/* Initiate selection */
A_W( 113, 3, ydelta_ret.Y );
A_W( 113, 2, ydelta_ret.Delta );
}
```

4.2.12 SpinCtrl, spindle control

Function description

The SpinCtrl function can be used to control spindles from the PLC. The function supports the functions:

- Position spindle
- Rotate spindle
- Oscillate spindle

The SpinCtrl function calls up basic function block FC18. For a description of FC18 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-25 Parameters for SpinCtrl

Signal	Type	Value range	Remarks
Start	VKE_TYPE		Start spindle control from PLC
Stop	VKE_TYPE		Stop spindle control from PLC
Funct	UBYTE	1, 2, 3	1: Position spindle 2: Rotate spindle 3: Oscillate spindle
Mode	UBYTE	0..5	0: Pos at absolute pos. 1: Pos incremental 2: Pos shortest path 3: Pos absolute, positive approach 4: Pos absolute, negative approach 5: Direction of rotation as M4
AxisNo	UWORD	1..8	No. of axis to be traversed
Pos	FLOAT	± 0.1469368 E -38 to ± 0.1701412 E +39	Rotary axis: degrees
FRate	FLOAT	± 0.1469368 E -38 to ± 0.1701412 E +39	Rotary axis: degrees/min Spindle: rpm

Return parameters

Table 4-26 Return parameters for SpinCtrl

Signal	Type	Value range	Remarks
InPos	VKE_TYPE		1 = Position reached or fct. executed
Error	VKE_TYPE		1 = error
State	UBYTE	0..255	Error code

Error detection

If a job could not be executed, this is indicated by a ‘1’ in the state parameter. The cause of the error is coded in the State block output:

Table 4-27 Error codes for SpinCtrl

State	Meaning
Error caused by PLC handling:	
1	Several functions were activated simultaneously for axis/spindle
20	A function was started although the position was not reached
Error caused by NCK handling: The alarm numbers are described in the 840D Diagnostics Guide.	
100	corresponds to alarm number 16830
105	corresponds to alarm number 16770
106	corresponds to alarm number 22052
107	corresponds to alarm number 22051
108	corresponds to alarm number 22050
109	corresponds to alarm number 22051
115	Programmed position was not reached
System or other major alarms:	
200	corresponds to system alarm number 450007

Programming example

```
/* Programming example for SpinCtrl */

VOID Bsp_SpinCtrl
{
SPINCTRL_STAT spinctrl_ret;

spinctrl_ret = SpinCtrl(   E_R( 76, 1 ),/*Start*/
                           E_R( 76, 0 ),/* Stop */
                           1,           /* Funct */
                           1,           /* Mode */
                           4,           /* AxisNo */
                           100.0,       /* Pos */
                           9.0    );    /* FRate */

A_W( 113, 5, spinctrl_ret.InPos );
A_W( 113, 4, spinctrl_ret.Error );
MW_W( 114, spinctrl_ret.State );
}
```

4.2.13 MCP_IFM, transfer of MCP signals to the interface

Function description

The MCP_IFM function (M variant) can be used to transfer operating modes, axis selections, WCS/MCS switchover, travel keys, overrides and keyswitch from the machine control panel (MCP) to the corresponding signals on the NCK/PLC interface. In the basic program, the handwheel selections, operating modes and other operating signals are transferred from the operator panel (OP) or MMC to the NCK/PLC interface, such that it is possible to select the operating modes from either the MCP or the OP.

The MCP_IFM function calls up basic function block FC19. For a description of FC19 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-28 Parameters for MCP_IFM

Signal	Type	Value range	Remarks
BAGNo	UBYTE	0..1	No. of mode group to which the operating mode signals are transferred.
ChanNo	UBYTE	0..2	Channel number for the channel signals.
SpindleIFNo	UBYTE	0..8	Number of the axis interface declared as a spindle.

Return parameters

Table 4-29 Return parameters for MCP_IFM

Signal	Type	Value range	Remarks
FeedHold	VKE_TYPE		Feed hold from MCP, modal
SpindleHold	VKE_TYPE		Spindle hold from MCP, modal

Programming example

```
/* Programming example for MCP_IFM */

VOID Bsp_MCP_IFM
{
MCP_IFM_STAT        ifm_ret;

ifm_ret = MCP_IFM(  1,            /* ModeGrpNo */
                    1,            /* ChanNo */
                    4      );     /* SpindleIFNo */

/* Signals to LEDs */
A_W( 104, 0, ifm_ret.FeedHold );
A_W( 104, 1, ifm_ret.SpindleHold );
}
```

4.2.14 MCP_IFT, transfer of MCP/OP signals to the interface

Function description

The MCP_IFT function (T variant) can be used to transfer operating modes, the direction keys of 4 axes, WCS/MCS switchover, travel keys, overrides and keyswitch from the machine control panel (MCP) to the corresponding signals on the NCK/PLC interface. In the basic program, the handwheel selections, operating modes and other operating signals are transferred from the operator panel (OP) or MMC to the NCK/PLC interface, such that it is possible to select the operating modes from either the MCP or the OP. The MCP_IFT function calls up basic function block FC25. For a description of FC25 please refer to

References: /PLCGP/, Description of Functions: Standard Machine

Parameters

Table 4-30 Parameters for MCP_IFT

Signal	Type	Value range	Remarks
BAGNo	UBYTE	0..1	No. of mode group to which the operating mode signals are transferred.
ChanNo	UBYTE	0..2	Channel number for the channel signals.
SpindleIFNo	UBYTE	0..8	Number of the axis interface declared as a spindle.

Return parameters

Table 4-31 Return parameters for MCP_IFT

Signal	Type	Value range	Remarks
FeedHold	VKE_TYPE		Feed hold from MCP, modal
SpindleHold	VKE_TYPE		Spindle hold from MCP, modal

Programming example

```
/* Programming example for MCP_IFT */

VOID Bsp_MCP_IFT
{
MCP_IFT_STAT        ift_ret;

ift_ret = MCP_IFT(  1,            /* ModeGrpNo */
                    1,            /* ChanNo */
                    4      );     /* SpindleIFNo */

/* Signals to LEDs */
A_W( 104, 0, ift_ret.FeedHold );
A_W( 104, 1, ift_ret.SpindleHold );
}
```

Miscellaneous 5

5.1 Access to local data from the C program

In order to access the start information of the standard OBs from the C program, the start information of the OBs is copied to communication data block DB70 on every OB entry.

Table 5-1 Detailed description of OB start information (extract from standard OB shells):

C function name	Standard OB	Start information in DB70
StdApplCycle()	OB 1	DBB 80 to 99
StdTimeAlert()	OB 10	DBB 100 to 119
StdDelayedTimeAlert()	OB 20	DBB 120 to 139
StdWatchdogAlert()	OB 35	DBB 140 to 159
StdProcessAlert()	OB 40	DBB 160 to 189
StdApplStart()	OB 100	DBB 190 to 209

OB 1

```
OB1_EV_CLASS: byte;               //Bits 0-3 = 1 (Coming event)
                                  //Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1: byte;                 //1(Cold restart scan 1 of OB 1)
                                  //3 (Scan 2-n of OB 1)
OB1_PRIORITY: byte;               //1(Priority of 1 is lowest)
OB1_OB_NUMBR: byte;               //1(Organization block 1, OB1)
OB1_RESERVED_1: byte;             //Reserved for system
OB1_RESERVED_2: byte;             //Reserved for system
OB1_PREV_CYCLE: int;              //Cycle time of previous OB1 scan
                                  (milliseconds)
OB1_MIN_CYCLE: int;               //Minimum cycle time of OB1
                                  //(milliseconds)
OB1_MAX_CYCLE: int;               //Maximum cycle time of OB1
                                  (milliseconds)
OB1_DATE_TIME: date_and_time;     //Date and time OB1 started
```

OB 10

```
OB10_EV_CLASS : byte;             //Bits 0-3=1 (Coming event)
                                  //Bits 4-7 = 1 (Event class 1)
OB10_STRT_INFO : byte;            //16#11 (OB 10 has started)
OB10_PRIORITY : byte;             //2 (Priority of 1 is lowest)
OB10_OB_NUMBR : byte;             //10 (Organization block 10, OB10)
OB10_RESERVED_1 : byte;           //Reserved for system
OB10_RESERVED_2 : byte;           //Reserved for system
OB10_PERIOD_EXE : word;           //Period of execution (once, per
                                  //minute/hour/day/week/month/year)
OB10_RESERVED_3 : int;//Reserved for system
OB10_RESERVED_4 : int;//Reserved for system
OB10 DATE_TIME : date_and_time;            //Date and time OB10 started
```

OB 20

```
OB20_EV_CLASS : byte;          //Bits 0-3 =1 (Coming event)
                               //Bits 4-7=1 (Event class 1)
OB20_STRT_INF : byte;          //16#21 (OB 20 has started)
OB20_PRIORITY : byte;          //3 (Priority of 1 is lowest)
OB20_OB_NUMBR : byte;          //20 (Organization block 20, OB20)
OB20_RESERVED_1 : byte;        //Reserved for system
OB20_RESERVED_2 : byte;        //Reserved for system
OB20_SIGN : word;              //Identifier input (SIGN)
                               //attached to SRT_DALM
OB20_RESERVED_3 : int;//Reserved for system
OB20_DTIME : int;              //Delay time (DTIME)
                               //input to SRT_DALM instruction
OB20_DATE_TIME: date_and_time; //Date and time OB20 started
```

OB 35

```
OB35_EV_CLASS : byte;          //Bits 0-3=1 (Coming event)
                               //Bits 4-7 = 1 (Event class 1)
OB35_STRT_INF : byte;          //16#36 (OB 35 has started)
OB35_PRIORITY : byte;          //11 (Priority of 1 is lowest)
OB35_OB_NUMBR : byte;          //35 (Organization block 35, OB35)
OB35_RESERVED_1 : byte;        //Reserved for system
OB35_RESERVED_2 : byte;        //Reserved for system
OB35_PHASE_OFFSET : word;      //Phase offset (msec)
OB35_RESERVED_3 : int;//Reserved for system
OB35_EXC_FREQ : int;           //Frequency of execution
                               //(XX integer) * (100 msec)
OB35_DATE_TIME: date_and_time; //Date and time OB1 started
```

OB 40

```
OB40_EV_CLASS:  byte;          //Bits 0-3=1 (Coming event)
                               //Bits 4-7 = 1 (Event class 1)
OB40_STRT_INF: byte;           //16#41 (OB 40 has started)
OB40_PRIORITY: byte;           //16 (Priority of 1 is lowest)
OB40_OB_NUMBR: byte;           //40 (Organization block 40, OB40)
OB40_RESERVED_1: byte;         //Reserved for system
OB40_MDL_ID: byte;             //Module identifier initiating interupt
OB40_MDL_ADDR: int;            //Base address of module initiating
OB40_POINT_ADDR: dword;        //Point address of interupt
OB40_DATE_TIME: date_and_time;//Date and time OB1 started
```

```
GP_IRFromNCK: bool;                //Interrupt for user by NCK
GP_TM: bool;                       //Tool management
GP_InPosition: array[1..31] of bool; //Axis-oriented for positioning axes,
                                   //Indexing axes, spindles
GP_AuxFunction: array[1..10] of bool;//Channel-oriented for miscellaneous
                                   //functions
GP_FMBlock: array[1..10] of bool;  //Channel-oriented for block transfer to FM
                                   //(in preparation)
```

OB 100

```
OB100_EV_CLASS : byte;             //16#13, Event class 1
                                   //Entering event state
                                   //Event logged in diag. buffer
OB100_STRTUP : byte;               //16#81/82/83/84 Method of startup
OB100_PRIORITY : byte;             //27 (Priority of 1 is lowest)
OB100_OB_NUMBR : byte;             //100 (Organization block 100, OB100)
OB100_RESERVED_1 : byte;           //Reserved for system
OB100_RESERVED_2 : byte;           //Reserved for system
OB100_STOP : word;                 //Event that caused CPU to stop (16#4xxx)
OB100_RESERVED_3 : word;           //Reserved for system
OB100_RESERVED_4 : word;           //Reserved for system
OB100_DATE_TIME: date_and_time; //Date and time OB100 started
```

5.2 Response to errors

The C block processing program supplies a return value, which is entered in memory words, to the S7 level issuing the call. When an error is detected, the PLC is switched to the STOP operating state.

Entry of return values

Tabelle 5-2 Entry of return values

S7 runtime level	Return value
OB 1	MW 40
OB10	MW 42
OB 20	MW 44
OB 35	MW 46
OB 40	MW 48
OB 100 C user block	MW 50
Monitor block for the HITEX monitor	MW 52

Possible return values

Table 5-3 Possible return values

Return value	Error	Remedy
0	No error	
1	Block call number too high	Use number between 0 and 14
2	Block does not exist	Match call number to block call number in generating file
3	DB_pointer does not exist	Generate and load DB_pointer
4	DB_pointer too small	Check entries in DB_pointer
5	Incorrect number of C blocks	"
6	Length block-spec. data incorrect (DB_pointer)	"
7	Definition in DB_pointer missing	"
8	Offset block-spec. data incorrect (DB_pointer)	"
9	DB_K missing	Generate and load DB_K
10	DB_K too small	Check DB_pointer
12	DB_K and DB_pointer identical	Assign different DB numbers
128	Monitor not loaded	Link in monitor during generation process
129	Monitor not initialized	Initialize monitor
130	DB_K for monitor not correct	Check DB_pointer
131 133	Incorrect C block number for debugger	Start monitor only for existing blocks with call number 0..14
132	Monitor called for several blocks	The monitor may only for activated for one C block at a time

Information about linker/locater error messages

Table 5-4 Linker/locater error messages

Error	Meaning
Warning 130	missing system stack definition This message is caused by the runtime system and has no significance.
Warning 152	class name '%s' not found Non-critical warning; if a class not contained in the current object is specified in the class instruction.
Warning 193	class '%s' without classes control Remedy: Include the designated section in the "classes" instruction for ROM or RAM or else the program response is not defined.
Error 268	'%s' linear element '%s %s' cannot be located within 4 pages Remedy: Check first whether the DPP assignments are correct (DPP formula = base address of specified data area/16k, e.g. 90000H/16k = 36)
Error 270	'%s' segmented element '%s %s' cannot be located The available memory has been exceeded, increase the specified memory limits for code or data section of locater.

Error handler for C library functions

The C library cs7rtlib.lno provides a freely programmable error handler *StdErrorHandler(), user_err.c* to which the program branches in the case of runtime errors. After processing, your application is continued. You can add application-specific error handling actions to the error handler.

The error codes for the detected errors can be found in header file *user_err.h.*

In the example project for rotary table positioning, the error handler (file *usr_err.c*) was extended insofar as the error code is entered in communication data block DB70.

• DW 40	Error code (error_id)
• DW 42	Errored parameter (error_reason_par)
• DD 44	Address of errored command (error_adr)

The PLC is also swiched to the STOP operating state in the example project for rotary table positioning. The user can now read out the cause of the error from the PLC status information (PG or MMC).

Example of an error entry in DB70:

DB70.DBW40 = h0431

DB70.DBW42 = h0080

DB70.DBD44 = h00080488

5.3 Stack handling on the PLC

The user stack is located in the near data segment of the C block. The C program requires this stack for the "automatic" variables as well as to manage the return addresses of function calls (-> C166 Special Stack Frame Library). The compiler calculates the stack size on the basis of the stack requirement of each individual function without the return addresses. This stack requirement is therefore based on the assumption that all functions are called simultaneously. In most cases, the requirement is oversized, but may be undersized in the case of a recursive call. The calculated stack requirement is occupied by runtime levels StdApplStart() (OB100) and StdApplCycle() (OB1) during the run time. A prespecified stack is located for each of the other runtime levels, i.e. this stack is not calculated by the compiler.

Modify constants for user stack

The user can modify the user stack by means of file *user_stk.c*. The stack is modified on the basis of the following constants:

```
OB1_STACK      LIT    '256'          ;StdApplCycle() & StdApplStart()
OB10_STACK     LIT    '512'          ;StdTimeAlert()
OB20_STACK     LIT    '512'          ;StdDelayedTimeAlert()
OB35_STACK     LIT    '512'          ;StdWatchdogAlert()
OB40_STACK     LIT    '512'          ;StdProcessAlert()
```

The value for OB1_STACK is an additive component in the C166 user stack calculation. This value acts as the default memory for the return addresses of function calls. Normally speaking, the user does not need to concern himself about the size of this stack. For description of the stack mechanism, please refer to:

References: /BSO/, Users Guide

5.4 Example project: Rotary table positioning

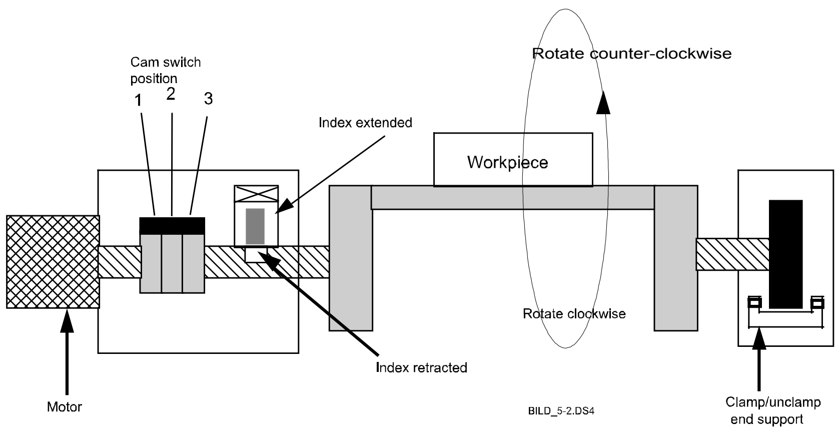


Fig. 5-1 Positioning a rotary table

Description of example

- The rotary table is driven by an electric motor with 2 speeds (rapid traverse and creep speed) and 2 directions (clockwise and counter-clockwise rotation).
- On approach to the position, the motor switches from rapid traverse to creep speed.
- Positioning is implemented by means of 3 cam switches with binary positional values 1, 2 and 4. Positions 1 to 7 are defined on the basis of these cam switches.
- The position reached is maintained by an index bolt. This bolt has 2 positions (retracted/extended) and is fixed by means of valves.
- The end support likewise has 2 positions (clamped/unclamped) and is valve-controlled. The support is unclamped during the traversing motion.
- The positioning operation is started by means of an expanded M function.
- The read-in disable is transferred to the NC during positioning.

Signal description

Table 5-5 Input signals for rotary table positioning example project

Input signal	Description
In_einfahren	Limit switch "Index retracted"
In_ausgefahren	Limit switch "Index extended"
Ge_geklemmt	Pressure switch "End support clamped"
Mo_inPosition	Limit switch "Motor (table) in position"
Mo_Cod1	Coding switch with positional value 1
Mo_Cod2	Coding switch with positional value 2
Mo_Cod4	Coding switch with positional value 4

Table 5-6 Output signals for rotary table positioning example project

Output signal	Description
In_einfahren	Retract index
In_ausfahren	Extend index
Ge_klemmen	Clamp end support
Ge_loesen	Unclamp end support
Mo_Rechts	Enable "Motor clockwise rotation"
Mo_Links	Enable "Motor counter-clockwise rotation"
Mo_Schnell	Enable "Motor high speed"

Table 5-7 Interface signals for rotary table positioning example project

Interface signals	Description
M_Funktion	M function for rotary table e.g.: M1=90 (90 = rotary table)
M_Adresse	M address for position e.g.: M1=90 (1 = Position)
Einlesesperre_an_NC	Read-in disable to NC

Description of operational sequence

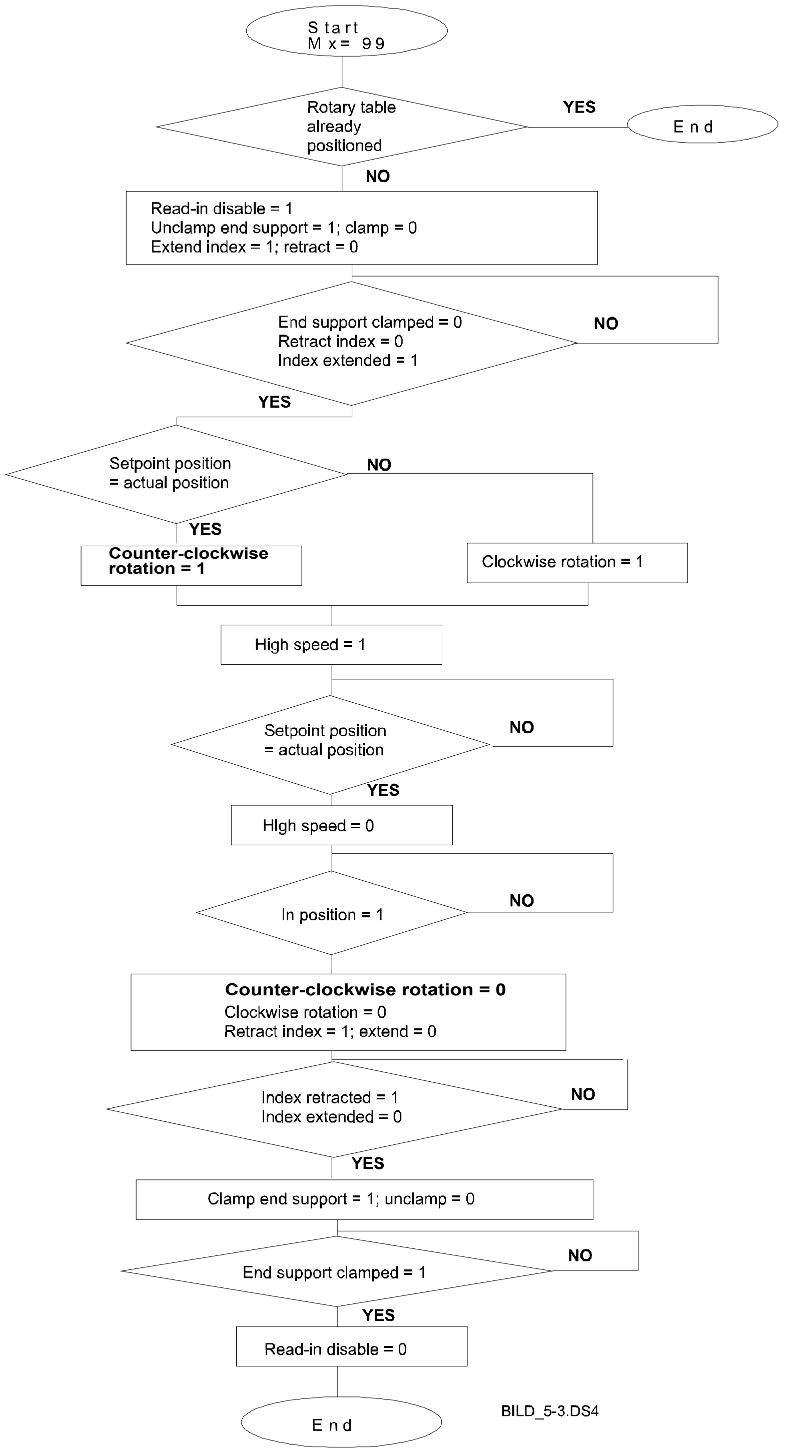


Fig. 5-2 Flowchart

Abbreviations

AB	Assembler block
AS314	Automation System 314
AS315	Automation System 315
DB	Data Block (SIMATIC S7)
FB	Function Block (SIMATIC S7)
FC	Function Call (SIMATIC S7)
MMC	Man-Machine Communication
MPI	Multipoint Interface
NCK	NC Kernel
OB	Organization Block (SIMATIC S7)
PII	Process Input Image
PIQ	Process Output Image
PLC	Programmable Logic Controller
SDB	System Data Block (SIMATIC S7)
STL	Statement List

References

B

/BSO/	80C166 Assembler Users Guide Vol I,II 80C166 C-Cross Compiler Users Guide 1993 Tasking Software B.V.
/MMC0/	SINUMERIK FM-NC/SINUMERIK 840D Operator's Guide 1994 SIEMENS AG
/PLCGP/	SINUMERIK FM-NC/SINUMERIK 840D Description of Functions, Basic Machine (Part 1), P3: Basic PLC Program 1994 SIEMENS AG
/BC3/	Borland C++3.1 C++ Development Package
/S7/	SIMATIC STEP 7 USER MANUAL 1994 SIEMENS AG
/HITEX/	TELEMON 167 User Manual HITEX System Development Gesellschaft für angewandte Informatik mbH, D-76229 Karlsruhe, Germany

Index C

SIEMENS AG AUT V24 P.O. Box 3180 D-91050 Erlangen Fed. Rep. of Germany	**Suggestions** **Corrections** For Publication/Manual: SINUMERIK 840D C-PLC Programming Manufacturer Documentation
From Name: Company/department Address: ______________________ ______________________ Telephone: ________ / ______________ Telefax: ______ / ________________	Description of Functions Order No.: 6FC5297-3AB60 Edition: 03.96 Should you come across any printing errors when reading this publication, please notify us on this sheet. Suggestions for improvements are also welcome.

Suggestions and/or corrections

Siemens AG
Automation Group
Automation Systems
for Machine Tools, Robots
and Special-Purpose Machines
P.O. Box 3180, D - 91050 Erlangen
Federal Republic of Germany

Siemens Aktiengesellschaft

Order No.: 6FC5297-3AB60
Printed in the Federal Republic of Germany