SIEMENS

SIMATIC - NET

SPC3 and DPS2
User Description

(Siemens PROFIBUS Controller
according to IEC 61158)

Version: 1.0
Date: 2003/04/09

Liability Exclusion
We have tested the contents of this document regarding agreement with the hardware and software described. Nevertheless, there may be deviations, and we don't guarantee complete agreement. The data in the document is tested periodically, however. Required corrections are included in subsequent versions. We gratefully accept suggestions for improvement

Subject to technical changes.

Revisions

Release	Date	Changes
V 1.0	2003/04/09	Only the user interface is described here. HW details can be found in the SPC3 HW description. Optimized max TSDR included
V 1.1	2009/04/29	Addresses

Directory

1 Introduction

For simple and fast digital exchange between programmable logic controllers, Siemens offers its users several ASICs. These ASICs are based on and are completely handled on the principles of the IEC 61158, of data traffic between individual programmable logic controller stations.
The following ASICs are available to support intelligent slave solutions, that is, implementations with a microprocessor.

The **ASPC2** already has integrated many parts of Layer 2, but the **ASPC2** also requires a processor's support. This ASIC supports baud rates up to 12 Mbaud. In its complexity, this ASIC is conceived primarily for master applications.

Due to the integration of the complete PROFIBUS-DP protocol, the **SPC3** decisively relieves the processor of an intelligent PROFIBUS slave. The **SPC3** can be operated on the bus with a baud rate of up to 12 MBaud.

However, there are also simple devices in the automation engineering area, such as switches and thermoelements, that do not require a microprocessor to record their states.

There are two additional ASICs available with the designations **SPM2** (Siemens Profibus Multiplexer, Version 2) and **LSPM2** (Lean Siemens PROFIBUS Multiplexer) for an economical adaptation of these devices. These blocks work as a DP slave in the bus system and work with baud rates up to 12 Mbaud. A master addresses these blocks by means of Layer 2 of the 7 layer model. After these blocks have received an error-free telegram, they independently generate the required response telegrams.

The LSPM2 has the same functions as the SPM2, but the LSPM2 has a decreased number of I/O ports and diagnostics ports.

2 Function Overview

The SPC3 makes it possible to have a price-optimized configuration of intelligent PROFIBUS-DP slave applications.

The processor interface supports the following processors:

Intel:	80C31, 80X86
Siemens:	80C166/165/167
Motorola:	HC11-,HC16-,HC916 types

In SPC3 the complete DP slave protocol is integrated. The DPS2 software realized a simple to use software interface to the user program.

The **integrated 1.5k Dual-Port-RAM** serves as an interface between the SPC3 and the software/application. The entire memory is subdivided into 192 segments, with 8 bytes each. Addressing from the user takes place directly and from the internal microsequencer (MS) by means of the so-alled base pointer. The base-pointer can be positioned at any segment in the memory. Therefore, all buffers must always be located at the beginning of a segment.

If the SPC3 carries out a DP communication the SPC3 automatically sets up all DP-SAPs. The various telegram information is made available to the user in separate data buffers (for example, parameter setting data and configuration data). Three change buffers are provided for data communication, both for the output data and for the input data. A change buffer is always available for communication. Therefore, no resource problems can occur. For optimal diagnostics support, SPC3 has two diagnostics change buffers into which the user inputs the updated diagnostics data. One diagnostics buffer is always assigned to SPC3 in this process.

The **bus interface** is a parameterizable synchronous/asynchronous 8-bit interface for various Intel and Motorola microcontrollers/processors. The user can directly access the internal 1.5k RAM or the parameter latches via the 11-bit address bus.

After the processor has been switched on, procedural-specific parameters (station address, control bits, etc.) must be transferred to the **Parameter Register File** and to the **mode registers**.

The *MAC status* can be scanned at any time in the **status register**.

Various events (various indications, error events, etc.) are entered in the **interrupt controller**. These events can be individually enabled via a mask register. Acknowledgement takes place by means of the acknowledge register. The SPC3 has a common interrupt output.

The integrated **Watchdog Timer** is operated in three different states: ‘Baud_Search’, ‘Baud_Control,’ and ‘DP_Control’.

The **Micro Sequencer (MS)** controls the entire process.

Procedure-specific parameters (buffer pointer, buffer lengths, station address, etc.) and the data buffer are contained in the integrated **1.5kByte RAM** that a controller operates as Dual-Port-RAM.

In **UART,** the parallel data flow is converted into the serial data flow, or vice-versa. The SPC3 is capable of automatically identifying the baud rates (9.6 kBd - 12 MBd).

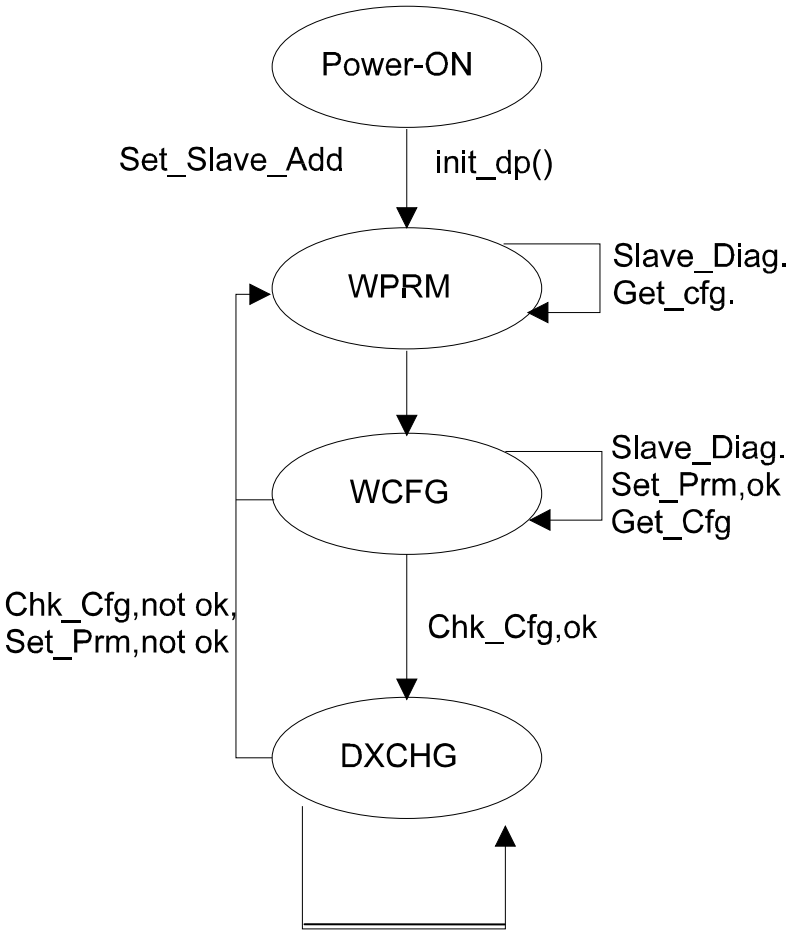
The **Idle Timer** directly controls the bus times on the serial bus cable.

3 PROFIBUS Development Kit

When purchasing a development kit you also will receive the licence of the DPS2 software for the cyclic data exchange. The DPSE software can be purchased separately. In this case we will only charge the update fee afterwards. The development kit consists of a ROFIBUS DP Master board (IM 180) as an ISA- board, the complete bus wiring and 2 slave boards (Order no.. 6ES7 195-3BA00-0YA0).

3.1 State Machine of a PROFIBUS DP Slave

For the sake of clarity, the state machine of a DP slave will be briefly described below. The detailed description is found in the IEC 61158.



DataExchange,OK
Rd_inp,
Rd_Outp,
Commands (Sync, Freeze...)
Slave_Diag,
Chk_Cfg,OK,
Set_Prm,OK,
Get_Cfg

The sequence in principle of this state machine is helpful in understanding the firmware sequence. Details are found in the Standard.

3.1.1 Power On

A Set_Slave_Address is accepted only in the Power_On state.

3.1.2 Wait_Prm

After start-up, the slave expects a parameter assignment message. All other types of messages are rejected or not processed. Data exchange is not yet possible.

At least the information specified by the Standard, such as PNO Ident Number, Sync-Freeze capability etc. is stored in the parameter message. In addition, user-specific parameter data is possible. Only the application specifies the meaning of this data. For example, certain bits are set to indicate a desired measuring range in the master interface configuration. The firmware makes this user-specific data available to the application program. The application program evaluates and accepts the data, but can also reject it (for example, the desired measuring range can't be set, and therefore meaningful operation isn't possible).

3.1.3 Wait_Cfg

The configuration message specifies the number of input bytes and output bytes. The master tells the slave how many bytes I/O are transferred. The application is notified of the requested configuration for verification. This verification either results in a correct, an incorrect, or an adaptable configuration. If the slave wants to adapt to the desired configuration, a new user data length has to be calculated from the configuration bytes (for example, 4 bytes I pre-defined and only 3 bytes utilized). The application has to decide whether this adaptability makes sense.

In addition, it is possible to query each master for the configuration of any slave.

3.1.4 Data_Exchange

If the firmware as well as the application have accepted the parameter assignment and the configuration as correct, the slave will enter the Data_Exchange state; that is, the slave exchanges user data with the master.

3.1.5 Diagnostics

The slave notifies the master of its current state by means of diagnostics. This state consists at least of the information specified in the Standard in the first six octets, as, for example, the status of the state machine. The user can supplement this information with process-specific information (user diagnostics, such as wire break).

On the slave's initiative, the diagnostics can be transmitted as an error message and as a status message. In addition to the three defined bits, the user also influences the application-specific diagnostics data. However, any master (not only the assigned master) can query the current diagnostics information.

- > Please note the detailed diagnostics description in the Appendix !

3.1.6 Read_Inputs, Read_Outputs

Any slave (in the Data_Exchange state) can query any master about the current states of the inputs and outputs. The ASIC and the firmware process this function autonomously.

3.1.7 Watchdog

Along with the parameter message, the slave also receives a watchdog value. If the bus traffic does not retrigger this watchdog, the state machine will enter the „safe" state Wait_Prm.

3.2 Optimizations of the bus cycle

For optimizations of the bus cycle the following adjustments of the max TSDR timings can be done in the GSD file.

Transmission rate (kbit/s)	187,5	500	1500	3000	6000	12000
Optimized Max TSDR	15	15	25	50	100	200

4 Overview DPS 2

4.1 Introduction

The PROFIBUS DP ASIC SPC3 almost completely relieves a connected microprocessor of processing the PROFIBUS DP state machine. The PROFIBUS DP ASIC SPC3 has functions permanently integrated in the internal microprogram, which in the case of earlier ASICs had to be carried out by the associated firmware.

The interface to the user is the register or RAM interface, which is to be located in the hardware description.

The DPS2 program package for the SPC3 relieves the SPC3 user of hardware register manipulations and memory calculations. DPS2 provides a convenient „C"-interface, and particularly provides support when the buffer organization is set up. For the SPC2, a transition from DPS2 to DPS2/SPC3 is simple, since the call-ups and the organization are the same.

The entire project package consists of:

Module		Function
userspc3.c	Main Program	The following functions are serviced here: start-up, input/output, and diagnostics
intspc3.c	Interrupt Module	This module handles the following functions: parameter assignment and configuration
dps2spc3.c	Help Functions	These functions calculate the buffer organization from the desired configuration.
dps2user.h	Macros and Definitions	These macros make it simple for the user to access the ASIC register structure.

As an interface to the user, DPS2 needs an interrupt for the SPC3 that the user must set up. The functions which have to be carried out when the ASIC interrupt occurs are included in the intspc3.c program.

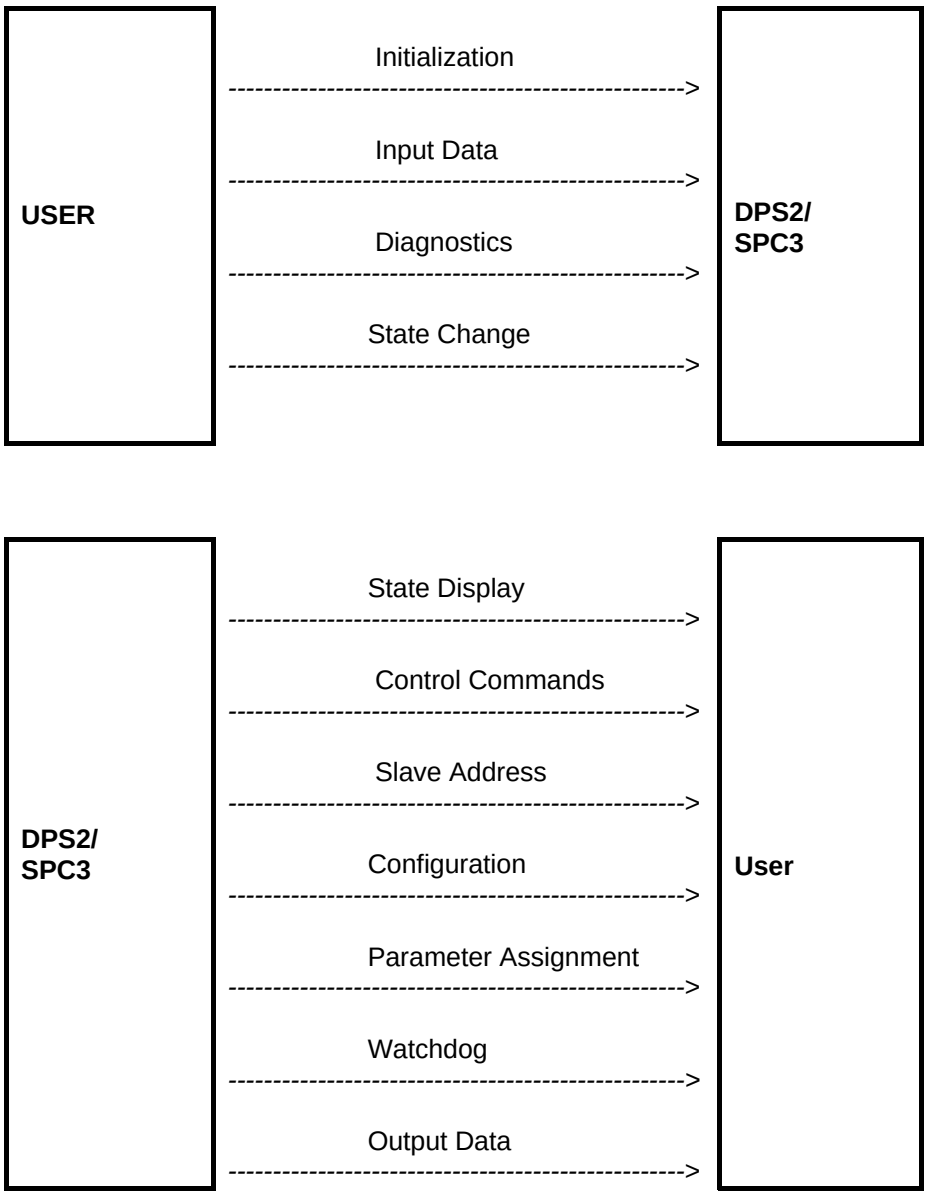
The user program can block this interrupt temporarily. It is also possible to block the interrupt entirely and process the corresponding functions with the polling process.

The interface between the user and the DPS2 firmware is divided into sequences and functions:

- Which the application makes available and which DPS2 calls up,

and functions

- Which DPS2 makes available and which the DPS2 application calls up.



4.2 Initialization

4.2.1 Hardware

During the first start-up step, the application program resets the ASIC SPC3 via the RESET pin, initializes the internal RAM and the resets connections of the connected processor.

4.2.2 Compiler Settings

The SPC3_INTEL_MODE literal sets the representation of the word registers in the SPC3.

The _INTEL_COMP literalsets the swap mechanism of the macros; that is, swapping bytes in a word.

SPC3_INTEL_MODE/_INTEL_COMP		
Transfer	#define	Intel Interface of the SPC3 selected
	not defined	Motorola Interface of the SPC3 selected
Return	------	

Processor	Compiler	Settings	Comment
SAB 165	Boston Tasking	SPC3_INTEL_MODE _INTEL_COMP	
80C32	Keil Compiler	SPC3_INTEL_MODE	Compiler represents word sizes in Motorola format => the swap mechanism of the macros has to be activated.

With the declaration #define DPS2_SPC3 the DPS2 interface is activated.

To support the different memory allocation models the accesses to the SPC3 are distinguished with a seperate attribute.
For C166-Compiler the addressing range of the SPC3 is as follows
#define SPC3_NEAR /* SPC3 is addressed in the NEAR-range*/
#define SPC3_FAR /* the SPC3 is addressed in the FAR range */

For 80C32-Compiler the addressing of the user data is as follows
#define SPC3_DATA_XDATA /* user data is located to the external RAM*/
#define SPC3_DATA_IDATA /* user data is located to the internal RAM*/

With the definition #define SPC3_NO_BASE_TYPES the declaration of the basic types (UBYTE, BYTE, UWORD, WORD) can be suppressed.

4.2.3 Locating the SPC 3

To have an easy access at the SPC3 it is possible to define a structure with the type SPC3. It has to be located at the address range defined by the hardware.

4.2.4 Hardware Mode

The macro DPS2_SET_HW_MODE (|) makes various SPC3 settings possible.

DPS2_SET_HW_MODE(x)	Hardware Settings	
Transfer		
	INT_POL_LOW	The interrupt output is low active.
	INT_POL_HIGH	The interrupt output is high active.
	EARLY_RDY	Ready is moved ahead by one pulse.
	SYNC_SUPPORTED	Sync_Mode is supported.
	FREEZE_SUPPORTED	Freeze_Mode is supported.
	DP_MODE	DP_Mode is enabled; the SPC3 sets up all DP_SAPs.
	EOI_TIMEBASE_1u	The interrupt inactive time is at least 1 usec.
	EOI_TIMEBASE_1m	The interrupt inactive time is at least 1 ms
	USER_TIMEBASE_1m	The User_Time_Clock interrupt occurs every 1 ms.
	USER_TIMEBASE_10m	The User_Time_Clock interrupt occurs every 10 ms. Describe again in more detail!
	SPEC_CLEAR	The SPC3 has to accept failsave-telegramms
Return	------	

The User_Time_Clock is a timer freely available for the application. This timer generates a 1 ms and a 10 ms timer tick. Through a relevant enable, this timer tick leads to an interrupt. (Refer to the following paragraph.)

4.2.5 Activating the Indication Function

The DPS2_SET_IND (|) macro activates the indication functions and interrupt triggers. The transfer parameters can be represented as UWORD, as BYTE (ending _B) and as BIT (ending: _NR).

DPS2_SET_IND(x|x..)	Activate Indication Field	
Transfer here UWORD Representa-tion	MAC_RESET	After processing the current job, the SPC3 has entered the *Offline State* by setting the 'Go_Offline' bit.
	GO_LEAVE_DATA_EX	The DP_SM has entered the 'DATA_EX' state or has exited it.
	BAUDRATE_DETECT	The SPC3 has exited the 'Baud_Search State' and has found a baud rate.
	WD_DP_MODE_TIMEOUT	The watchdog timer has expired in the 'DP_Control' WD state.
	USER_TIMER_CLOCK	The time base of the User_Timer_Clock has expired (1/10ms) timer tick.
	Reserved	for additional functions
	Reserved	for additional functions
	Reserved	for additional functions
	NEW_GC_COMMAND	The SPC3 has received a 'Global_Control Message' with a changed 'GC_Command-Byte' and has stored this byte in the 'R_GC_Command' RAM cell.
	NEW_SSA_DATA	The SPC3 has received a 'Set_Slave_Address Message' and has made the data available in the SSA buffer.
	NEW_CFG_DATA	The SPC3 has received a 'Check_Cfg Message' and has made the data available in the Cfg buffer.
	NEW_PRM_DATA	The SPC3 has received a 'Set_Param Message' and has made the data available in the Prm buffer.
	DIAG_BUFFER_CHANGED	On request by 'New_Diag_Cmd', the SPC3 has exchanged the diagnostics buffers and has made the old buffer available again to the user.
	DX_OUT	The SPC3 has received a 'Write_Read_Data Message' and has made the new output data available in the N buffer. For 'Power_On' or for 'Leave_Master', the SPC3 clears the N buffer contents and also generates this interrupt.
	Reserved	For additional functions
	Reserved	For additional functions
Return	------	

Example:

DPS2_SET_IND(GO_LEAVE_DATA_EX | WD_DP_MODE_TIMEOUT);
*/ The user is informed when the DATA_Exchange state is entered or exited, or when the watchdog timer has run out. */

An interrupt activation with byte variables could look like this:

DPS2_SET_IND(NEW_CFG_DATA_B | NEW_PRM_DATA_B | USER_TIMER_CLOCK_B);

4.2.6 User Watchdog

The user watchdog ensures that if the connected microprocessor fails, the SPC3 leaves the data cycle after a defined number (DPS2_SET_USER_WD_VALUE) of data messages. As long as the microprocessor doesn't „crash", it has to retrigger this watchdog (DPS2_RESET_USER_WD).

DPS2_SET_USER_WD_VALUE (x)		Set User Watchdog Time
Transfer	UWORD	Number of data messages
Return	------	

DPS2_RESET_USER_WD()		Complete restart / retriggering of user watchdog
Transfer	------	
Return	------	

In the worst case scenario, the data telegrams can be sent in the time interval of the Min_Slave interval. By means of this time specification and the run length of its own program component, the application can specify the number of data messages.

Sample calculation: ($T_{application\ runtime}$ / min_slave interval) x 2 = number of data telegrams

Refer to DIN E 19245 Part 3 (maximum master polling time of telegrams to the slave).
2 = safety factor

4.2.7 Station Address

During startup, the application program reads in the station address (DIL switch, EEPROM, etc.), and transfers the station address to the ASIC. The user must also specify whether this station address can be changed via the PROFIBUS DP; that is, a memory medium (for example, serial EEPROM) is available.

DPS2_SET_STATION_ADRESS (x)		Set Station Address
Transfer	UBYTE	Address
Return	------	

DPS2_SET_ADD_CHG_DISABLE()		Station Address Change Disabled
Transfer	------	
Return	------	

DPS2_SET_ADD_CHG_ENABLE()		Station Address Change Permitted Attention: The user must set up buffers for this utility!
Transfer	------	
Return	------	

4.2.8 Ident Number

During startup, the application program reads in the ident number (EPROM, host system) and transfers it to the ASIC.

DPS2_SET_IDENT_NUMBER_HIGH(x)		Ident Number
Transfer	UBYTE	High byte of PNO ident number
Return	------	

DPS2_SET_IDENT_NUMBER_LOW(x)		Ident Number
Transfer	UBYTE	Low byte of PNO ident number
Return	------	

4.2.9 Response Time

If special circumstances require it, the user can set the response time for the SPC3 during set-up. In operation with PROFIBUS DP, the parameter message of the PROFIBUS DP master specifies the response time.

DPS2_SET_MINTSDR(x)		MinTsdr
Transfer	UBYTE	Response time in bit timing (11-255)
Return	------	

4.2.10 Buffer Initialization

The user must enter the lengths of the exchange buffers for the different messages in the dps2_buf structure of the DPS2_BUFINIT type. These lengths determine the data buffers set up in the ASIC, and therefore are dependent in total sum on the ASIC memory. DPS2_INIT checks the maximum lengths of the buffers entered, and returns the test result. Please specify the overall calculation. Is the in/out buffer mutually specified?

```
typedef struct {
  UBYTE din_dout_buf_len;     /*overall length of the input/output buffer, 0-488*/
  UBYTE diag_buf_len;         /*length of the diagnostics buffer, 6-244*/
  UBYTE prm_buf_len;          /*length of the parameter buffer, 7-244*/
  UBYTE cfg_buf_len;          /*length of the config data buffer, 1-244*/
  UBYTE ssa_buf_len;          /*length of the Set-Slave-Add buffer, 0 and 4-244*/
} DPS2_BUFINIT;
```

Specifying the length 0 for the Set-Slave-Address buffer disables this utility.

For this type of buffer initialization, an additional macro is needed for adapting the lengths of the Din/Dout buffers, since these are the only ones that are allowed to be changed during operation (but not beyond the preset size).

DPS2_INIT (x)	Buffer Initialization	
Transfer	Pointer to values with the DPS2_BUFINIT structure	Desired/required buffer lengths
Return	DPS2_INITF_DIN_DOUT_LEN	Error with Din/Dout length
	DPS2_INITF_DIAG_LEN	Error with diagnostics length
	DPS2_INITF_PRM_LEN	Error with parameter assignment data length
	DPS2_INITF_SSA_LEN	Error with address data length
	DPS2_INITF_LESS_LEN	Overall, too much memory used
	DPS2_INITF_OK	Buffer length OK

4.2.11 Entry of Setpoint Configuration

With the macro, the function first fetches a pointer to a data block for the configuration.

DPS2_GET_READ_CFG_BUF_PTR()	Fetch Pointer to Configuration Buffer	
Transfer	----	
Return	UBYTE *	Pointer to RAM area in the SPC3

In this data block, the user enters his configuration (identifier bytes). The individual identifier bytes are to be generated according to the following specification (refer also to IEC 61158):

Bit

7	6	5	4	3	2	1	0

Data Length 00 = Byte/Word
15 = 16Byte/Words

In- /Output 00 = Special Identifier Format
01 = Input
02 = Output
11 = Input - Output

Lengt h 0 = Byte, Byte Structure
1 = Word

Consistency across 0 = Byte or Word
1 = Total Length

For example, the identifiers correspond to 17 hex = 8 bytes input without consistency

27 hex = 8 bytes output without consistency

The special identifier formats are to be found in IEC 61158.

With the DPS2_SET_READ_CFG_LEN (CFG_LEN) macro, the user sets the length of the configuration data entered.

DPS2_SET_READ_CFG_LEN (x)		Set Length of Configuration Data
Transfer	UBYTE	Length of entries in the configuration buffer
Return	----	

Then the user uses the dps2_calculate_inp_outp_len() function made available in the dps2spc3.c file to determine the length of the input and output data from the identifier bytes. This function returns a pointer to a structure of the DPS2_IO_DATA_LEN type. A zero pointer indicates a faulty buffer configuration (for example, real_cfg_data_len = 0).

dps2_calculate_inp_outp_len(x,y)		Calculation of Inputs/Outputs
Transfer	UBYTE *	Pointer to configuration buffer
	UWORD	Length of configuration data
Return	DPS2_IO_DATA_LEN *	Pointer to structure with the calculated input- output lengths

```
typedef struct {
UBYTE inp_data_len;
UBYTE outp_data_len;
} DPS2_IO_DATA_LEN;
```

With the DPS2_SET_IO_DATA_LEN(ptr) macro, the user initiates the DPS2 variables inp_data_len and outp_data_len.

DPS2_SET_IO_DATA_LEN(x)		Set Input-/Output Data Lengths
Transfer	DPS2_IO_DATA_LEN *	Pointer to structure with the calculated input-/output lengths
Return	UBYTE	TRUE: sufficient memory available FALSE: memory insufficient

4.2.12 Fetching the First Buffer Pointers

Before the first entry of its input data, the application has to fetch a buffer for the input data with the DPS2_GET_DIN_BUF_PTR() macro. With the DPS2_INPUT_UPDATE() macro, the user can transfer the input data to DPS2. The length of the inputs is not transferred with every input; the length must agree with the length transferred by DPS2_SET_IO_DATA_LEN().

Macro DPS2_GET_DIN_BUF_PTR()		Fetch First Input Data Buffer
Transfer	-----	
Return	UBYTE *	Pointer to input buffer

Before the first entry of external diagnostics, the user must get a pointer to the available diagnostics buffer with the DPS2_GET_DIAG_BUF_PTR() macro. The user can then enter his diagnostics messages or status messages (starting with Byte 6) in this buffer.

DPS2_GET_DIAG_BUF_PTR()		Fetch first diagnostics buffer.
Transfer	-----	
Return	UBYTE *	Pointer to diagnostics buffer; NIL if no diagnostics buffer available anymore

4.2.13 Baudrate Control

With the DPS2_SET_BAUD_CNTRL () macro, the root value of baudrate monitoring can be set. After the set time (Value x Value x 10ms), the SPC3 autonomously starts the baudrate search, if no valid message was received during this time. If the master system uses the watchdog, the value the master specified for baud rate monitoring is used for watchdog monitoring. If the slave is operated without a watchdog, ASIC

SPC3 interprets the entry of the root value for the baud rate monitoring. This makes a time value in the range of 10 ms - 650 s possible (entry 2-255).

DPS2_SET_BAUD_CNTRL (x)		Baudrate Monitoring
Transfer	UBYTE	Root value of baudrate monitoring
Return	------	

4.2.14 Start of the SPC3

With DPS2_START, the SPC3 switches itself on-line.

DPS2_START ()		Start SPC3
Transfer	------	
Return	------	

4.3 DPS2 Interface Functions

4.3.1 DPS2 Indication Function (dps2_ind())

The user has to set up and make the dps2_ind() interrupt function ready. DPS2 will carry out this function as soon as a corresponding event has occurred which was enabled in the interrupt bit field with the DPS2_SET_IND() macro. (See above.)

dps2_ind		Interrupt Function
Transfer	-------	
Return	-------	

In a 16-bit field, the DPS2 indicates the reason for the indication to the user with bits, on which literals have been entered.

4.3.2 Read Out Reason for Indication

With the DPS2_GET_INDICATION macro, the user receives the event which has caused the indication, the interrupt trigger.

DPS2_GET_INDICATION()		Read Out Reason for Indication
Transfer	--------	
Return	UWORD	Refer to the field described under DPS2_SET_IND

In order to increase the performance, primarily the 803x and 805x (byte-oriented), you can also query each indication with its own macro (DPS2_GET_IND_...) instead. A runtime-optimized interface can be created with these macros.

DPS2_GET_IND_GO_LEAVE_DATA_EX()	The DP_SM has entered the 'DATA_EX' state or has exited it.
DPS2_GET_IND_MAC_RESET()	After processing the current request, the SPC3 has entered the *offline state* (by setting the 'Go_Offline' bit).
DPS2_GET_IND_BAUDRATE_DETECT()	The SPC3 has left the 'Baud_Search state' and has found a baud rate.
DPS2_GET_IND_WD_DP_MODE_TIMEOUT	In the 'DP_Control' WD state , the watchdog timer has expired.
DPS2_GET_IND_USER_TIMER_CLOCK	The time base of the User_Timer_Clock has expired (1/10ms).
DPS2_GET_IND_NEW_GC_COMMAND()	The SPC3 has received a 'Global_Control Message' with a changed 'GC_Command Byte' and has stored this byte in the 'R_GC_Command' RAM cell.
DPS2_GET_IND_NEW_SSA_DATA()	The SPC3 has received 'Set_Slave_Address Message' and has made the data available in the SSA buffer.
DPS2_GET_IND_NEW_CFG_DATA()	The SPC3 has received Check_Cfg Message' and has made the data available in the Cfg buffer.
DPS2_GET_IND_NEW_PRM_DATA()	The SPC3 has received 'Set_Param Message' and has made the data available in the Prm buffer.
DPS2_GET_IND_DIAG_BUFFER_CHANGED()	Requested by 'New_Diag_Cmd' , the SPC3 has exchanged the diagnostics buffer and has made the old buffer available again to the user.
DPS2_GET_IND_ DX_OUT()	The SPC3 has received a 'Write_Read_Data Message' and has made the new output data available in the N buffer. For 'Power_On' and for 'Leave_Master', the SPC3 clears the N buffer contents and also generates this interrupt.

Transfer	--------	
Return	UBYTE	0/FALSE: no interrupt 1/TRUE: This indication/interrupt has occurred.

4.3.3 Acknowledging the Indication

The DPS2_IND_CONFIRM() macro acknowledges the indication received through dps2_ind().

DPS2_IND_CONFIRM(x)		Acknowledge the Indication
Transfer	UWORD	Refer to the field described under DPS2_SET_IND.
Return	--------	

Performance can also be increased by here defining a macro each for each indication (see „Read Out the Reason for indication“).

DPS2_CON_IND_GO_LEAVE_DATA_EX()	See above
DPS2_CON_IND_MAC_RESET()	
DPS2_CON_IND_BAUDRATE_DETECT()	
DPS2_CON_IND_WD_DP_MODE_TIMEOUT	
DPS2_CON_IND_USER_TIMER_CLOCK	
DPS2_CON_IND_NEW_GC_COMMAND()	
DPS2_CON_IND_NEW_SSA_DATA()	
DPS2_CON_IND_NEW_CFG_DATA()	
DPS2_CON_IND_NEW_PRM_DATA()	
DPS2_CON_IND_DIAG_BUFFER_CHANGED()	
DPS2_CON_IND_ DX_OUT()	

Transfer	--------	
Return	--------	

4.3.4 Ending the Indication

The DPS2_SET_EOI() macro ends the indication sequence / interrupt function.

DPS2_SET_EOI()		Close Interrupt
Transfer	------	
Return	------	

4.3.5 Polling the Indication

The user can also poll indications instead of having them signaled with dps2_ind(). The DPS2_POLL_IND_xx macro is available for a single read-out, or DPS2_POLL_INDICATION() for global read-out. Polled indications can likewise be acknowledged with the DPS2_IND_CONFIRM macro.

DPS2_POLL_INDICATION()		Reason for Indication
Transfer	--------	
Return	UWORD	Refer to the field described under DPS2_SET_IND.

DPS2_POLL_IND_GO_LEAVE_DATA_EX()	The DP_SM has entered the 'DATA_EX' state or has exited it.
DPS2_POLL_IND_MAC_RESET()	After processing the current request, the SPC3 has entered the o*ffline state* (by setting the 'Go_Offline' bit
DPS2_POLL_IND_BAUDRATE_DETECT()	The SPC3 has left the 'Baud_Search State' and found a baud rate.
DPS2_POLL_IND_WD_DP_MODE_TIMEOUT()	In the WD state 'DP_Control', the watchdog timer has expired.
DPS2_POLL_IND_USER_TIMER_CLOCK()	The time base of the User_Timer_Clock has expired (1/10ms).
DPS2_POLL_IND_NEW_GC_COMMAND()	The SPC3 has received a 'Global_Control Message' with a changed 'GC_Command-Byte' and has filed this byte in the 'R_GC_Command' RAM cell .
DPS2_POLL_IND_NEW_SSA_DATA()	The SPC3 has received a 'Set_Slave_Address Message' and has made the data available in the SSA buffer.
DPS2_POLL_IND_NEW_CFG_DATA()	The SPC3 has received a 'Check_Cfg Message' and has made the data available in the Cfg buffer.
DPS2_POLL_IND_NEW_PRM_DATA()	The SPC3 has received a 'Set_Param Message' and has made the data available in the Prm buffer.
DPS2_POLL_IND_DIAG_BUFFER_CHANGED()	Requested by 'New_Diag_Cmd', the SPC3 has exchanged the diagnostics buffers and made the old buffer available again to the user.
DPS2_POLL_IND_ DX_OUT()	The SPC3 has received a 'Write_Read_Data Message' and has made the new output data available in the N buffer. For 'Power_On' and for 'Leave_Master', the SPC3 clears the N buffer and also generates this interrupt.

Transfer	--------	
Return	UBYTE	0/FALSE: No interrupt 1/TRUE: This indication/interrupt has occurred.

4.3.6 Checking Parametrization

The user has to program the function for checking the received parameter assignment data. DPS2 calls up the dps2_ind function in which NEW_PRM_DATA can determine whether the checking function has to be carried out. Macro call-ups from DPS2 can fetch the required pointer to the corresponding buffer and the length of this buffer.

The DPS2_GET_PRM_LEN() macro determines the length of the received data.

DPS2_GET_PRM_LEN ()		Fetch parameter buffer length.
Transfer	--------	
Return	UBYTE	Length of the parameter data buffer

DPS2_GET_PRM_BUF_PTR() supplies a pointer to the current parameter buffer.

DPS2_GET_PRM_ BUF_PTR()		Fetch pointer to parameter buffer.
Transfer	--------	
Return	UBYTE *	Address of the parameter buffer

Within this verification function, the user has the task of checking the received User_Prm_Data for validity. The user acknowledges the checked parameters as positive by calling the DPS2_SET_PRM_DATA_OK macro, and as negative by calling DPS2_SET_PRM_DATA_NOT_OK(). By acknowledging with these macros, the interrupt request is canceled; that is, this interrupt may **no** longer be acknowledged with DPS2_IND_CONFIRM(). The return value of the macros has to be evaluated as described below.

DPS2_SET_PRM_DATA_OK()		The received parameter assignment is OK.
DPS2_SET_PRM_DATA_NOT_OK()		This macro notifies DPS2 the parameter assignment isn't OK. The transferred parameters can't be used in the device.
Transfer	--------	
Return	DPS2_PRM_FINISHED	No further parameter assignment message is present => end of sequence.
	DPS2_PRM_CONFLICT	Another parameter assignment message is present! => repeat check of requested parameter assignment.
	DPS2_PRM_NOT_ALLOWED	Access in present bus mode is not permitted. For example, it is possible the watchdog has run out during verification. Verifying the parameter setting data (and possibly series-connected functions in the application) are to be cancelled.

Caution:

When configuration settings and parameter settings are received, first there **must** be verification of the **parameter setting data** and their confirmation. Then the configuration settings must be verified. The sequence is absolutely mandatory.

4.3.7 Checking Configuration Data

The user has to program the function for verifying received configuration data. DPS2 calls up the dps2_ind function in which NEW_CFG_DATA can determine whether the verification function has to be carried out. Macro calls from DPS2 supply the needed pointer as well as the buffer length.

The DPS2_GET_CFG_LEN() macro determines the length of the received data.

DPS2_GET_CFG_LEN ()		Fetch configuration buffer length.
Transfer	--------	
Return	UBYTE	Length of the received configuration byte

DPS2_GET_CFG_BUF_PTR() supplies a pointer to the current configuration buffer.

DPS2_GET_CFG_ BUF_PTR()		Fetch pointer to configuration buffer.
Transfer	--------	
Return	UBYTE *	Configuration buffer address

Within the verification function, the user has the task of comparing the received Cfg_Data with the Real_Cfg_Data; that is, its possible configuration. The user acknowledges the verified configuration data as positive by calling up the macro DPS2_SET_CFG_DATA_OK() or DPS2_SET_CFG_DATA_UPDATE(). The usre acknowledges the verified configuration data as negative by calling up DPS2_SET_CFG_DATA_NOT_OK() negative. By acknowledging with these macros, the interrupt request is removed; that is, this interrupt may **no** longer be acknowledged through DPS2_IND_CONFIRM(). The return value of the macros has to be evaluated as described below.

DPS2_SET_CFG_DATA_OK()	The transferred configuration is OK.
DPS2_SET_CFG_DATA_UPDATE()	If the user desires the verified configuration be exchanged with the one already in DPS2, this can be done with the *DPS2_SET_CFG_DATA_UPDATE()* macro.
DPS2_SET_CFG_DATA_NOT_OK()	This macro notifies the DPS2 that the configuration is not OK.

Transfer	--------	
Return	DPS2_CFG_FINISHED	No further configuration message is present => end of sequence.
	DPS2_CFG_CONFLICT	An additional configuration message is present! => Repeat verification of the requested configuration.
	DPS2_CFG_NOT_ALLOWED	Access is not permitted in the present bus mode. For example, it is possible the watchdog has run out during verification. The verification of the configuration data (and possibly subsequent functions in the application) are to be cancelled.

4.3.8 Transfer of Output Data

DX_OUT in dps2_ind() displays received output data. The macro DPS2_OUTPUT_UPDATE() changes the output buffers.

The DPS2_OUTPUT_UPDATE_STATE() buffer supplies the buffer pointer, and also the state of the Dout buffer.

The lengths of the outputs are not transferred with every update. The length agrees with the length transferred with DPS2_SET_IO_DATA_LEN(). If this were not the case, DPS2 would return to the WAIT-PRM state.

DPS2_OUTPUT_UPDATE_STATE ()		Fetch buffer pointer and state of the output buffer.
Transfer	UBYTE *	Pointer to variable into which the state of the output buffer is to be written
Return	UBYTE *	Pointer to output data buffer

The following states (bits) are encoded into the status (pointer to this variable was transferred):

NEW_DOUT_BUF	Received new output data
DOUT_BUF_CLEARED	Output data was deleted.

DPS2_OUTPUT_UPDATE ()		Fetch buffer pointer to output buffer.
Transfer	------	
Return	UBYTE *	Pointer to output buffer or NIL, if no buffer

4.3.9 Transfer of Input Data

As described, the application has to fetch a buffer for the input data with the DPS2_GET_DIN_BUF_PTR() macro before the first entry of its input data.

With the DPS2_INPUT_UPDATE() macro, the user can repeatedly transfer the current input data from the user to DPS2. The length of the inputs is not transferred with every update.. The length must agree with the length transferred by DPS2_SET_IO_DATA_LEN().

DPS2_INPUT_UPDATE ()		Fetch buffer pointer to input buffer.
Transfer	------	
Return	UBYTE *	Pointer to input data buffer

The input-/output data length can be reconfigured with the functions and macros described in the "Initialization" section (dps2_calculate_inp_outp_len(), DPS2_SET_IO_DATA_LEN(), ...).

4.3.10 Transferring Diagnostics Data

With this utility, the user can transfer diagnostics data to DPS2. Prior to the first entry of external diagnostics data, the user has to get a pointer to the free diagnostics buffer with the DPS2_GET_DIAG_BUF_PTR() macro. The user can then write his diagnostics messages or status messages (starting with Byte 6) into this buffer.

DPS2_GET_DIAG_BUF_PTR()		Fetch pointer to diagnostics data buffer.
Transfer	------	
Return	UBYTE *	Pointer to diagnostics buffer NIL if no diagnostics data buffer in the 'U' state

The user specifies the length of the diagnostics data by calling up the DPS2_SET_DIAG_LEN() macro. The length is only to be set after a buffer was successfully received with DPS2_GET_DIAG_BUF_PTR().

The length **always** has to be transferred for the entire buffer, including the bytes specified by the standard (+6). This means that, if no user diagnostics is supposed to be transferred, the **length 6** is to be transferred.

DPS2_SET_DIAG_LEN()		Set length of diagnostics data.
Transfer	UBYTE	Length of diagnostics data
Return	UBYTE	Diagnostics length actually set 0xff, if no buffer is assigned to the user

The transferred pointer of DPS2 points to Byte 0 of the transferred diagnostics buffer. The user may enter his diagnostics in this buffer starting with **Byte 6**. DPS2 enters the fixed diagnostics bytes (bytes 0 to 5).

Structure of the data block to be transferred for expanded diagnostics:

Byte	Diagnostics Data	Comment
0	Station Status_1	Byte 0 to 5 permanent diagnostics header
1	Station Status_2	
2	Station Status_3	
3	Diag.Master_Add	
4	Ident_Number_High	
5	Ident_Number_Low	
6 to 241 max.	Ext_Diag_Data	Start of user diagnostics in the DP Standard format

With the DPS2_S ET_DIAG_STATE() macro, the user transfers the new diagnostics state to DPS2. The new diagnostics state has to be transferred before the diagnostics data is updated.

DPS2_SET_DIAG_STATE()			Setting the Diagnostics Bits
Transfer	Bit	Designation	Meaning
	0	EXT_DIAG	If this bit is 1, the diagnostics bit Diag.Ext_Diag will be set; otherwise, the bit will be reset.
	1	STAT_DIAG	If this bit is 1, the diagnostics bit Diag.Stat_Diag will be set; otherwise, the bit will be reset.
	2	EXT_DIAG_OVF	If this bit is 1, the bit Diag.Ext_Diag_Overflow is set; otherwise, Diag.Ext_Diag_Overflow is reset.
Return	------		

With the DPS2_DIAG_UPDATE() macro, the user transfers the new, external diagnostics data to DPS2. As a return value, the user receives a pointer to the new diagnostics data buffer.

DPS2_DIAG_UPDATE()		Transfer diagnostics data and fetch new pointer.
Transfer	------	
Return	UBYTE *	Pointer to the diagnostics buffer; NIL if no diagnostics data buffer present

If no diagnostics data is to be transferred with the DPS2_DIAG_UPDATE() macro, or if the diagnostics data transferred previously is to be deleted, the diagnostics length has to be set to 6 with the DPS2_SET_DIAG_LEN() macro. The SPC3 responds to a diagnostics request from the PROFIBUS DP master with the 6 bytes of station diagnostics data.

4.3.11 Checking Diagnostics Data Buffers

The other exchange buffer is not automatically available after the diagnostics data has been transferred. The user has two possibilities to find out when the diagnostics buffer was transmitted:

- DPS2 signals via the dps2_ind() indication function and indicates the event with DIAG_BUFFER_CHANGED. This indication function has to be enabled during initialization for this purpose.

With the DPS2_GET_DIAG_FLAG() macro, the user polls the state of the diagnostics buffer. The macro indicates whether the buffer has already been transmitted. If, however, „static diagnostics“ has been set, the „buffer not
transmitted“ state is always returned.

DPS2_GET_DIAG_FLAG()		Fetch state of diagnostics buffer.
Transfer	------	
Return	UBYTE	TRUE: Diagnostics buffer has not yet been transmitted (or static diagnostics). FALSE: Diagnostics buffer has already been transmitted.

4.3.12 Changing the Slave Address

NEW_SSA_DATA indicates a request to change in the slave address. With the DPS2_GET_SSA_BUF_PTR() macro, a pointer to the buffer with the new slave address can be determined, and with DPS2_GET_SSA_LEN() macro, the length of the received SSA buffer can be determined.

DPS2_GET_SSA_LEN()		Length of the Set_Slave_Address Buffer
Transfer	------	
Return	UBYTE	Length of the SSA buffer

DPS2_GET_SSA_BUF_PTR()		Fetch Pointer of Set_Slave_Address Buffer.
Transfer	------	
Return	UBYTE *	SSA buffer address

The user has to acknowledge the transfer of the data by calling the DPS2_SET_SSA_BUF_FREE() macro.

DPS2_SET_SSA_BUF_FREE()		Acknowledging the Set_Slave_Address utility
Transfer	------	
Return	------	

4.3.13 Signaling Control Commands

This message signals the arrival of a Global_Control message. The message is only made if group association and a change of the control command was recognized as compared to the previous command. The DPS2_GET_GC_COMMAND() macro supplies the Control_Command byte. This makes it possible for the user to additionally react to these commands. The DPS2 internally processes these commands regarding buffer management. That is, in the case of „Clear“, the output data is deleted.

DPS2_ GET_GC_COMMAND ()			Fetch Global Control Command
Transfer	----		
Return	Bit	Designation	Meaning
	0	Reserved	
	1	Clear_Data	This command deletes the output data and makes the data available to the user. A switch to 'U' is made.
	2	Unfreeze	With „Unfreeze", the freeze of input data is canceled.
	3	Freeze	The input data is „frozen." The application does not fetch new input data until the master sends the next „freeze" command.
	4	Unsync	The „Unsync" command cancels the „Sync" command.
	5	Sync	The output data last received is made available to the application. The following transferred output data is not passed on to the application until the next 'Sync' command is given.
	6,7	Reserved	The „Reserved" designation indicates that these bits are reserved for future function expansions.

4.3.14 Leaving the Data Exchange State

The GO_LEAVE_DATA_EX message indicates that DPS2 has carried out a state change of the internal state machine.

With the DPS2_GET_DP_STATE() macro, the application is informed whether the DPS2 has entered the data exchange state or left it. The cause for this can be a faulty parameter assignment message in the data transfer phase, for example.

DPS2_GET_DP_STATE():		Fetching the status of the PROFIBUS DP state machine
Transfer	------	
Return	DPS2_DP_STATE_WAIT_PRM	Wait for parameter assignment
	DPS2_DP_STATE_WAIT_CFG	Wait for configuration
	DPS2_DP_STATE_DATA_EX	Data exchange
	DPS2_DP_STATE_ERROR	Error

4.3.15 DPS2_Reset (Go_Offline)

With this macro, the SPC3 enters the offline state. The offline state can only be exited with the DPS2_INIT function. This provides the possibility to transfer and start new configuration data.

DPS2_RESET()		Go to the offline state.
Transfer	-------	
Return	-------	

The DPS2_GET_OFF_PASS() macro can help to determine whether the transition to offline was made.

DPS2_GET_OFF_PASS()		Check the offline state.
Transfer	--------	
Return	UBYTE/Bit	1 = Passive idle 0 = Offline

4.3.16 Response Monitoring Expired

WD_DP_MODE_TIMEOUT indicates the sequence of response monitoring. The SPC3_GET_WD_STATE() macro queries the status of the watchdog state machine.

SPC3_GET_WD_STATE()		State of the watchdog state machine
Transfer	------	
Return	SPC3_WD_STATE_BAUD_SEARCH	Baudr ate search
	SPC3_WD_STATE_BAUD_CONTROL	Checking the baudrate
	SPC3_WD_STATE_DP_MODE	DP_Mode; that is, bus watchdog activated

4.3.17 Requesting Reparameterization

The DPS2_USER_LEAVE_MASTER() macro causes the DPS2/SPC3 to change into the „Wait_Prm“ state.

DPS2_USER_LEAVE_MASTER()		Enter the State Wait_Prm
Transfer	------	
Return	------	

4.3.18 Reading Out the Baudrate

The DPS2_GET_BAUD() macro supplies the recognized baud rate in coded form.

DPS2_GET_BAUD()		Read baud rate.
Transfer	--------	
Return	BD_12M	12 MBaud
	BD_6M	6 MBaud
	BD_3M	3 MBaud
	BD_1_5M	1.5 MBaud
	BD_500k	500 KBaud
	BD_187_5k	187.5 KBaud
	BD_93_75k	93.75 KBaud
	BD_19_2k	19.2 KBaud
	BD_9_6k	9.6 KBaud

4.3.19 Determining Addressing Errors

The SPC3 indicates MAC_RESET and ACCESS_VIOLATION when an addressing error occurs during an access above 1.5 KB of the internal RAM. The macros SPC3_GET_OFF_PASS() and SPC3_GET_ACCESS_VIOLATION() are provided to distinguish between the transition between "offline" and "passive" when an addressing error occurs.

SPC3_GET_ACCESS_VIOLATION()		Addressing error has occurred
Transfer	------	
Return	UBYTES	≠ 0: Addressing error occurred = 0: No addressing error

Caution:
In C32 mode, an erroneous access of the processor does not trigger an interrupt. An erroneous access of the SPC3's internal microsequencer does generate a message, however.

4.3.20 Determining the Free Memory Space in the SPC3

During initialization, the SPC3_INI() macro sets up buffer space in the internal RAM of the SPC3. You can use this macro to provide yourself with a pointer to the beginning of the free memory space in the SPC3, and the number of bytes still available. This functions returns a ZERO pointer when the SPC3 has not been initialized.

SPC3_GET_FREE_MEM()		Determine free memory space
Transfer	UBYTE *	Pointer to the location containing the memory space available
Return	UBYTE *	Pointer to the free memory space in the SPC3 0 when SPC3 was not initialized correctly

5 Sample Program

5.1 Overview

The sample program shows the utilization of the DPS2 software with the following examples:

- The received output data is filed in a defined memory area (io_byte_ptr).
- As input data, this memory area is read back or mirrored.
- The first byte of this input data influences the diagnostics bits in the manner already described.
- The sample slave has a switched on configuration of 0x13 / 0x23 (that is, 4 bytes I/Q) and can adapt itself to a configuration of 0x11/0x21 that is, 2 bytes I/Q). Based on your application, you must decide the extent to which a configuration change is a good idea
- If 0xAA and 0xAA is in the user-specific parameter data, the sample program will signal

a faulty parameter assignment. The user-specific parameter data is copied to the diagnostics data field.

You can insert your application to the interfaces described. The program modules to be processed are summarized in the user directory. You particularly have to determine and enter the station address via your mechanism (for example, rotary switch, keys, etc.). You can obtain your own device-/manufacturer-specific PNO ident number from the PNO (refer to address list). You can include your own interrupt programs, dependent on the application, in the interrupt routines provided in the source code.

Sample batch files, command files etc. are included in the diskette directory for generating operational EPROMs.

The current state is stored on the delivery diskette. Please heed the current implementation instructions in the interface center's mailbox (++49 911 73 79 72).

5.2 Main Program

The following sample program shows the principal sequence of DPS2 in an application.

Das folgende Beispielprogramm zeigt den prinzipiellen Ablauf von DPS2 in einer Anwendung.

```
/*****************************************************************************/
/*  D e s c r i p t i o n :                                                  */
/*                                                                           */
/*  USER-TASK                                                                */
/*****************************************************************************/


void    main ()
{
/* Reset sequenz for the SPC3 and the microprocessor         */
/*   depending of the used hardware application              */
/* - force the Reset Pin                                     */
/* - Set the interrupt parameters of the microprocessor      */
/* - Delete the SPC3 internal RAM */

/* activate the indication functions */
SPC3_SET_IND(GO_LEAVE_DATA_EX | WD_DP_MODE_TIMEOUT | NEW_GC_COMMAND |\
               NEW_SSA_DATA | NEW_CFG_DATA | NEW_PRM_DATA | BAUDRATE_DETECT);

/* set the watchdog value in the SPC3, which supervice the microprozessor */
DPS2_SET_USER_WD_VALUE(20000);


/* In this example the input and output bytes are transfered to the
   IO area, which is addressed by the io_byte_ptr. In the case of the IM183
   there is RAM. */


#ifdef _IM182
    io_byte_ptr = achIO;    //set memory adr.
#else
    io_byte_ptr = ((UBYTE*) 0x2E000L);
#endif
for (i=0; i<2; i++)
    {
    (*(io_byte_ptr + i)) = 0;
    }

/* fetch the station address, in this case the station address
   is fixed in EPROM*/
this_station = OWN_ADDRESS;

/* get the Identnumber    */
ident_numb_high = IDENT_HIGH;
ident_numb_low =  IDENT_LOW;

/* Allow the change of the slave address by the PROFIBUS DP */
real_no_add_chg = FALSE;

/* Allow not the change of the slave address by the PROFIBUS DP      */
/* Attention: The set_slave_address service is with it not blockaded */
real_no_add_chg = TRUE;


/* Reset the User und DPS */
user_dps_reset();


for (;;)
    {   /*=== Begin  of the endless loop ===*/
#ifdef _IM182
        if(kbhit())
        {
            break;
        }
  #ifndef PC_USE_INTERRUPT
        dps2_ind();
  #endif
#endif
    zyk_wd_state = SPC3_GET_WD_STATE();    /*for info.: the actuall WD State*/

    zyk_dps_state = DPS2_GET_DP_STATE();   /*for info.: the actuall PROFIBUS DP State*/


    DPS2_RESET_USER_WD();           /* Trigger the user watchdog of the SPC3 */

#ifdef __C51__
    HW_WATCHDOG_TRIGGER = 1;        /* Retrigger the HW Watchdog of the IM183*/
```

```
    HW_WATCHDOG_TRIGGER = 0;
#endif

/*============ Handling of the output data =================*/

    if (DPS2_POLL_IND_DX_OUT()) /* are new output date available? */
        {
        /* Confirm the taking over of the output data */
        DPS2_CON_IND_DX_OUT();

        /* Get the pointer to the actual output data */
        user_output_buffer_ptr = DPS2_OUTPUT_UPDATE();


        /* Example: Copy the output data to the IO */
        for (i=0; i<user_io_data_len_ptr->outp_data_len; i++)
           {
           (*((io_byte_ptr) + i)) = (*(((UBYTE SPC3_PTR_ATTR*) user_output_buffer_ptr) + i));
           }
        }


/*============ Handling of the input data =================*/

    /* Write the input data from the periphery to the ASIC */
    for (i=0; i<user_io_data_len_ptr->inp_data_len; i++)
        {
        *(((UBYTE SPC3_PTR_ATTR*) user_input_buffer_ptr) + i) = *((io_byte_ptr) + i);
        }

    /* Give the actual pointer / data to the SPC3/DPS2 an get a new pointer,
           where the next input data can be written */
    user_input_buffer_ptr = DPS2_INPUT_UPDATE();


/*== Handling of the external diagnosis and other user defined actions =====*/
/* ATTENTION:         this is only an example              */

/* Take the first Byte of the Input data as a service byte */
/*  for the change diag function                           */

    dps_chg_diag_srvc_byte_new = *((UBYTE*)(io_byte_ptr));

    if (user_diag_flag) /* is a diagnosis buffer available? */
        {
        /* Is there a change in the service byte (1.input byte) */
        if (dps_chg_diag_srvc_byte_new == dps_chg_diag_srvc_byte_old)
            {
             /* no action */
             }
        else
            {
            /*== Handling of the external diagnosis  =====*/
            /* only the least significant 3 byte are used */
            if ((dps_chg_diag_srvc_byte_new & 0x07) !=
                  (dps_chg_diag_srvc_byte_old & 0x07))
                {
                /* Mask the 3 bits */
                diag_service_code = dps_chg_diag_srvc_byte_new & 0x07;

                /* Write the length of the diagnosis data to the SPC3 */
                if (dps_chg_diag_srvc_byte_new & 0x01)
                    diag_len = 16;   //max. value of the IM308B
                else
                    diag_len = 6;
                diag_len = DPS2_SET_DIAG_LEN(diag_len);

                /* Write the external diagnosis data to the SPC3 */
                build_diag_data_blk ((struct diag_data_blk *)user_diag_buffer_ptr);

                /* Set the service code             */
                /* 0x01 External diagnosis          */
                /* 0x02 Static   diagnosis          */
                /* 0x04 External diagnosis Overflow */
                DPS2_SET_DIAG_STATE(diag_service_code);


                /* Trigger the diagnosis update in the SPC3*/
                DPS2_DIAG_UPDATE();

                /* Store "no diagnosis buffer available"  */
                user_diag_flag = FALSE;

                }

            dps_chg_diag_srvc_byte_old = dps_chg_diag_srvc_byte_new;

            }
        }
```

```
/*================ Check the buffers and the state =================*/

/* Is a new diagnosis buffer available */
     if (DPS2_POLL_IND_DIAG_BUFFER_CHANGED())
        {
        DPS2_CON_IND_DIAG_BUFFER_CHANGED(); /* Confirm the indication */
        user_diag_buffer_ptr = DPS2_GET_DIAG_BUF_PTR(); /* Fetch the pointer */
        user_diag_flag = TRUE; /* Set the Notice "Diag. buffer availble      */
        }

    }   /*=== endless loop        ===*/

#ifdef _IM182
#ifdef PC_USE_INTERRUPT
    if(uwPCIrq<8)
    {
        outp(PIC_MASTER + PIC_IMR, ubOldMask);
    }
    else
    {
        outp(PIC_SLAVE + PIC_IMR, ubOldMask);
    }
    _dos_setvect(uwPCInt, oldhandler);
#endif

    // force SPC3 to leave master
    outp(SPC3_RESET,0x21);
    outp(SPC3_RESET,0x00);
#endif
    return;

}



/*****************************************************************************/
/*  D e s c r i p t i o n :                                                  */
/*                                                                           */
/* Reset the USER and DPS                                                    */
/*****************************************************************************/


void user_dps_reset (void)
{
enum SPC3_INIT_RET dps2_init_result;            /* result of the initial. */


DPS2_SET_IDENT_NUMBER_HIGH(ident_numb_high);    /* Set the Identnumber    */
DPS2_SET_IDENT_NUMBER_LOW(ident_numb_low);

SPC3_SET_STATION_ADDRESS(this_station);         /* Set the station address*/

SPC3_SET_HW_MODE(SYNC_SUPPORTED | FREEZE_SUPPORTED | INT_POL_LOW | USER_TIMEBASE_10m);
                                                /* Set div. modes of the  */
                                                /* SPC3                   */
if (!real_no_add_chg)
    {
    DPS2_SET_ADD_CHG_ENABLE();                  /* Allow or allow not the */
    }                                           /* address change         */
else
    {
    DPS2_SET_ADD_CHG_DISABLE();
    }


/* initialize the length of the buffers for DPS2_INIT() */
dps2_buf.din_dout_buf_len = 244;
dps2_buf.diag_buf_len = sizeof(struct diag_data_blk);
dps2_buf.prm_buf_len = 20;
dps2_buf.cfg_buf_len = 10;

/* dps2_buf.ssa_buf_len = 5;    reserve buffer if address change is possible */
dps2_buf.ssa_buf_len = 0;       /* Suspend the address change service  */
                                /* No storage in the IM183 is possible */


/* initialize the buffers in the SPC3                   */
dps2_init_result = SPC3_INIT(&dps2_buf);
if(dps2_init_result != SPC3_INIT_OK)
    {       /* Failure */
    for(;;)
        {
        error_code = INIT_ERROR;
        user_error_function(error_code);
        }
    }
```

```
/* Get a buffer for the first configuration */
real_config_data_ptr = (UBYTE SPC3_PTR_ATTR*) DPS2_GET_READ_CFG_BUF_PTR();

/* Set the length of the configuration data */
DPS2_SET_READ_CFG_LEN(CFG_LEN);

/* Write the configuration bytes in the buffer */
*(real_config_data_ptr) = CONFIG_DATA_INP;      /* Example 0x13 */
*(real_config_data_ptr + 1) = CONFIG_DATA_OUTP; /* Example 0x23 */

/* Store the actuall configuration in RAM for the check in the
   check_configuration sequence (see the modul intspc3.c)       */
cfg_akt[0] = CONFIG_DATA_INP;
cfg_akt[1] = CONFIG_DATA_OUTP;
cfg_len_akt = 2;


/* Calculate the length of the input and output using the configuration bytes*/
user_io_data_len_ptr = dps2_calculate_inp_outp_len (real_config_data_ptr,(UWORD)CFG_LEN);
if (user_io_data_len_ptr != (DPS2_IO_DATA_LEN *)0)
    {
    /* Write the IO data length in the init block */
    DPS2_SET_IO_DATA_LEN(user_io_data_len_ptr);
    }
else
    {
    for(;;)
        {
        error_code =IO_LENGTH_ERROR;
        user_error_function(error_code);
        }
    }


/* Fetch the first input buffer */
user_input_buffer_ptr = DPS2_GET_DIN_BUF_PTR();

/* Fetch the first diagnosis buffer, initialize service bytes */
dps_chg_diag_srvc_byte_new = dps_chg_diag_srvc_byte_old = 0;
user_diag_buffer_ptr = DPS2_GET_DIAG_BUF_PTR();
user_diag_flag = TRUE;

/* for info: get the baudrate                */
user_baud_value = SPC3_GET_BAUD();

/* Set the Watchdog for the baudrate control */
SPC3_SET_BAUD_CNTRL(0x1E);

/* and finally, at last, los geht's start the SPC3 */
SPC3_START();

}
```

5.3 Interrupt Program

The following interrupt program shows the sequence in principle of the DPS2 interrupt program in an application.

```
/*****************************************************************************/
/*  D e s c r i p t i o n :                                                  */
/*                                                                           */
/*  dps2_ind                                                                 */
/*                                                                           */
/*  This function is called by the hardware interrupt                        */
/*****************************************************************************/

#if defined __C51__
    void dps2_ind(void)    interrupt 0
#elif   _C166
    interrupt (0x1b) void dps2_ind(void)    /* CC11 = EX3IN */
#else
    void dps2_ind(void)
#endif

{
UBYTE       i;

if(DPS2_GET_IND_GO_LEAVE_DATA_EX())
    {   /*=== Start or the end of the Data-Exchange-State ===*/
    go_leave_data_ex_function();
    DPS2_CON_IND_GO_LEAVE_DATA_EX();    /* confirm this indication */
    }

if(DPS2_GET_IND_NEW_GC_COMMAND())
    {   /*===  New Global Control Command ===*/
    global_ctrl_command_function();
    DPS2_CON_IND_NEW_GC_COMMAND();  /* confirm this indication */
    }

if(DPS2_GET_IND_NEW_PRM_DATA())
    {   /*=== New parameter  data ===*/
    UBYTE   SPC3_PTR_ATTR * prm_ptr;
    UBYTE   param_data_len, prm_result;
    UBYTE   ii;

    prm_result = DPS2_PRM_FINISHED;
    do
        { /* Check parameter until no conflict behavior */
        prm_ptr = DPS2_GET_PRM_BUF_PTR();
        param_data_len = DPS2_GET_PRM_LEN();

        /* data_length_netto of parametration_telegram > 7 */
        if (param_data_len > 7)
            {
            if (( *(prm_ptr+8) == 0xAA)  && ( *(prm_ptr+9) == 0xAA))
                prm_result = DPS2_SET_PRM_DATA_NOT_OK(); /* as example !!! */
            else
                {
                for (ii= 0; ii<param_data_len && ii <10; ii++)  // store in the interim buffer
                    prm_tst_buf[ii] = *(prm_ptr+ii+7);          // for the diagnostic
                                                          //!!!!!! as example !!!!

                prm_result = DPS2_SET_PRM_DATA_OK();
                }
            }
        else
            prm_result = DPS2_SET_PRM_DATA_OK();

        } while(prm_result == DPS2_PRM_CONFLICT);

    store_mintsdr =  *(prm_ptr+3);    // store the mintsdr for restart after
                                      // baudrate search

    }

if(DPS2_GET_IND_NEW_CFG_DATA())
    {   /*=== New Configuration data ===*/
    UBYTE SPC3_PTR_ATTR * cfg_ptr;
    UBYTE i, config_data_len, cfg_result, result;

    cfg_result = DPS2_CFG_FINISHED;
    result = DPS_CFG_OK;

    do
        {   /* check configuration data until no conflict behavior m*/
        cfg_ptr = DPS2_GET_CFG_BUF_PTR();             /* pointer to the config_data_block */
        config_data_len = DPS2_GET_CFG_LEN();
```

```
        /* In this example the only possible configurations are 0x13 and 0x23
           (4 Byte I/O) or 0x11 and 0x21 (2 Byte I/O) are possible */

        if ( config_data_len != 2)
            cfg_result = DPS2_SET_CFG_DATA_NOT_OK();
        else
            {   /* Length of the configuration data  o.k. */
                /* check the configuratin bytes           */

            if ((cfg_akt[0] == cfg_ptr[0]) && (cfg_akt[1] == cfg_ptr[1]))
                result = DPS_CFG_OK;
                /* the desired conf. is equal the actuall configuration */
            else
                {
                if (((cfg_ptr[0] == 0x13) && (cfg_ptr[1]) ==0x23)
                  || ((cfg_ptr[0] == 0x11) && (cfg_ptr[1]) ==0x21))
                    {
                    cfg_akt[0] = cfg_ptr[0];
                    cfg_akt[1] = cfg_ptr[1];
                    result = DPS_CFG_UPDATE;
                    }
                else
                    result = DPS_CFG_FAULT;              /* as example !!!!! */


                if (result == DPS_CFG_UPDATE)
                    {
                    user_io_data_len_ptr = dps2_calculate_inp_outp_len(
                        cfg_ptr,(UWORD)config_data_len);
                    if (user_io_data_len_ptr != (DPS2_IO_DATA_LEN *)0)
                        {
                        DPS2_SET_IO_DATA_LEN(user_io_data_len_ptr);
                        }
                    else
                        result = DPS_CFG_FAULT;
                    }
                }
            switch (result)
                {
                case DPS_CFG_OK: cfg_result = DPS2_SET_CFG_DATA_OK();
                    break;

                case DPS_CFG_FAULT: cfg_result = DPS2_SET_CFG_DATA_NOT_OK();
                    break;

                case DPS_CFG_UPDATE: cfg_result = DPS2_SET_CFG_DATA_UPDATE();
                    break;
                }
            }

        } while(cfg_result == DPS2_CFG_CONFLICT);
    }

if(DPS2_GET_IND_NEW_SSA_DATA())
    {   /*=== New Slave address received ===*/
    address_data_function(DPS2_GET_SSA_BUF_PTR(), DPS2_GET_SSA_LEN());
    DPS2_CON_IND_NEW_SSA_DATA();    /* confirm this indication */
    }

if(DPS2_GET_IND_WD_DP_MODE_TIMEOUT())
    {   /*=== Watchdog is run out ===*/
    wd_dp_mode_timeout_function();
    DPS2_CON_IND_WD_DP_MODE_TIMEOUT();  /* confirm this indication */
    }
if(SPC3_GET_IND_USER_TIMER_CLOCK())
    {   /*==== Timer tick received ====*/
    SPC3_CON_IND_USER_TIMER_CLOCK();
    }

if(SPC3_GET_IND_BAUDRATE_DETECT())
    {   /*==== Baudrate found  ====*/

    /* If the baudrate has lost and again found in the state WAIT_CFG,  */
    /*  DATA_EX the SPC3 would answer to the next telegramms            */
    /*  with his default mintsdr.                                       */
    /* But he should answer in the meantime parametrized mindstr        */

    if ((DPS2_GET_DP_STATE() ==  DPS2_DP_STATE_WAIT_CFG )
        ||  (DPS2_GET_DP_STATE()  ==  DPS2_DP_STATE_DATA_EX))
         SPC3_SET_MINTSDR(store_mintsdr);

    SPC3_CON_IND_BAUDRATE_DETECT();
    }
SPC3_SET_EOI(); /* */
}   /* End dps2_ind() */
```


6 Microcontroller Implementation

6.1 Developmental Environment

Keil C51-Compiler Version 4.01 or higher
Boston Tasking C165-Compiler

6.2 Diskette Contents

The hardware-dependent parts are shown as subfunctions in the sample program or in the other functions of the user directory.

Path	File	Description
user	userspc3.c	User program with main()
	intspc3.c	SPC3 interrupt (not in MINISPC3)
	dps2spc3.c	DPS2 help functions (not in MINISPC3)
	dps2user.h	Header file
lst		Directory for listings
obj	*.obj	Translate modules
	*.hex	Hex-file for EPROM
prj	us.bat	Compiler call-up for userspc3.c
	it.bat	Compiler call-up for intspc3.c (not in MINISPC3)
	d2.bat	Compiler call-up for dps2spc3.c (not in MINISPC3)
	link.bat	Linker/locator call
	spc3.l51	Linker command file
	spc3.log	Result file for linker-/locator run
	hex.bat	Call-up of the Object Hex Converter

6.3 Generation

You can translate and link the individual files in the user directory with the help of batches. Special note should be taken that the SPC3 will be located on the 0x1000 hardware address. If, through corresponding wiring, the SPC3 is placed on another address, the address instruction has to be adjusted, of course.

You can make adaptations to your hardware or your application in the respective files. The interrupt call-up interface and the operation of the pertinent control bits is available to you in the source code, so that you can insert your own procedures.

7 IM182 Implementation

7.1 Developmental Environment

The software was tested with following compilers:

- MSVC++ V 1.5
- Borland C/C++ V 4.0
- Watcom C/C++ V 10.0

The usage of other compilers should be possible without any problems.

7.2 Diskette Contents

The hardware-dependent parts are shown as subfunctions in the sample program or in the other functions of the user directory.

Path	File	Description
IM182	userspc3.c	User program with main()
	dps2spc3.c	DPS2 help functions (not in MINISPC3)
	spc3dps2.h	Header file
	spc3.ide	Projektfile für Borland Compiler
	spc3msvc.mak	Projektfile für Microsoft Compiler
	spc3wc.mak	Makefile for Watcom Compiler (16 bit DOS-Program)
	spc3wc3.mak	Makefile for Watcom Compiler (32 bit DOS4GW Program)

7.3 Generation

For Borland and Microsoft Compiler you can load the projectfile in the appropriate IDE and build the program.

!!! ATTENTION !!!
For the 32-bit DOS4GW variant you must define the macro SPC3_FLAT in the file SPC3DPS2.H (remove the comment).

8 Appendix

8.1 Addresses

PROFIBUS Nutzer Organisation
PNO
Office
Mr. Dr. Peter Wenzel
Haid- und Neu- Strasse 7
76131 Karlsruhe/Germany
Tel.: (0721) 9658-590

Contact Persons at the Interface Center in Germany

Siemens AG
Dept I IA SE DE DP3
Mr. Putschky
Würzburgerstr.121
90766 Fürth/Germany

Email:
gerd.putschky@siemens.com

Tel.: (0911) 750 - 2078
Fax: (0911) 750 - 2100

Contact Persons at the Interface Center in the USA

PROFIBUS Interface Center
One Internet Plaza
PO Box 4991
Johnson City, TN 37602-4991

Fax : (423) - 262 - 2103

Your Partner:
Tel.: (423) - 262 - 2576

Email:
profibus.sea@siemens.com

8.2 General Definition of Terms

ASPC2	Advanced Siemens PROFIBUS Controller, 2nd generation
SPC2	Siemens PROFIBUS Controller, 2nd generation
SPC3	Siemens PROFIBUS Controller, 3rd generation
SPM2	Siemens PROFIBUS Multiplexer, 2nd generation
LSPM2	Lean Siemens PROFIBUS Multiplexer, 2nd generation
DP	Distributed I/Os
FMS	Fieldbus Message Specification
MS	MicroSequenzer
SM	State Machine

9 Appendix A: Diagnostics Processing in PROFIBUS DP

9.1 Introduction

PROFIBUS DP offers a convenient and multi-layer possibility for processing diagnostics messages on the basis of error states.

As soon as a diagnostics request is required, the slave will respond in the current data exchange with a high priority reply message. In the next bus cycle, the master then requests a diagnostics from this slave, instead of executing normal data exchange.

Likewise, any master (not only the assigned master!) can request a diagnostics from the slave. The diagnostics information of the DP slave consists of standard diagnostics information (6 bytes), and can be supplemented by user-specific diagnostics information.

In the case of the ASICs, SPM2, and LSPM2, extensive diagnostics is possible through corresponding wiring. In the case of the intelligent SPCx solution, adapted and convenient diagnostics processing can be carried out through programming.

9.2 Diagnostics Bits and Expanded Diagnostics

Parts of the standard diagnostics information are permanently specified in the firmware and in the micro-program of the ASICs through the state machine.

Request diagnostics only once („update_diag(..)“) if an error is present or changes. By no means should diagnostics be requested cyclically in the data exchange state; otherwise, the system will be burdened by redundant data.

Three information bits can be influenced by the application:

9.2.1 STAT_DIAG

Because of a state in the application, the slave can't make valid data available. Consequently, the master only requests diagnostics information until this bit is removed again. The PROFIBUS DP state is, however, Data_Exchange, so that immediately after the cancellation of the static diagnostics, data exchange can start.

Example: failure of supply voltage for the output drivers

9.2.2 EXT_DIAG

If this bit is set, a diagnostics entry **must** be present in the user-specific diagnostics area. If this bit is not set, a status message can be present in the user-specific diagnostics area.

User-Specific Diagnostics

The user-specific diagnostics can be filed in three different formats:

Device-Specific Diagnostics:

The diagnostics information can be coded as required.

	Bit 7	Bit 6	Bit 5-0
Header Byte	**0**	**0**	Block length in bytes, including header
Diagnostics Field	Coding of diagnostics is device-specific		
.....	Can be specified as required		

Identifier-Related Diagnostics:

For each identifier byte assigned during configuration (for example, 0 x 10 for 1 byte input), a bit is reserved.

In the case of a modular system with an identifier byte each per module, module-specific diagnostics can be indicated. One bit respectively will then indicate diagnostics per module.

	Bit 7	Bit 6	Bit 5-0	
Header Byte	**0**	**1**	Block length in bytes including header	
Bit Structure	1			1

⇑ **Identifier Byte 7 has** **etc.** ⇑ **Identifier Byte 0 has**

diagnostics diagnostics

Channel-Related Diagnostics:

In this block, the diagnosed channels and the diagnostics cause are entered in sequence. Three bits are required per entry.

	Bit 7	Bit 6	Bit 5	Bit 4 - 0
Header Byte	**1**	**0**	Identification Number	
Channel Number	Coding Input/Output		Channel Number	
Type of Diagnostics	Coding Channel Type			Coding Error Type

Coding of the error type is in part manufacturer-specific; other codings are specified in the Standard.

Example:

0	**0**	0	0	0	1	0	0	**Device**-related diagnostics.
Device-specific								Meaning of the bits
diagnostics field of								is specified
length 3								manufacturer-specific.
0	**1**	0	0	0	1	0	1	**Identifier**-related diagnostics.
							1	Identification number 0 has diagnostics.
			1					
						1		Identification number 18 has diagnostics.
1	**0**	0	0	0	0	0	0	**Channel-**related diagnostics, identification number 0.
0	0	0	0	0	0	1	0	Channel 2.
0	0	0	1	0	1	0	0	Overload, channel organized bit by bit.
1	**0**	0	0	1	1	0	0	**Channel**-related diagnostics identification number 12.
0	0	0	0	0	1	1	0	Channel 6.
1	0	1	0	0	1	1	1	Upper limit evalue xceeded, channel organized word by word.

Status

If the Bit EXT_DIAG is set to 0 , data is viewed as status info from the system view. f.e. cancellation of the error triggering the diagnostics.

9.2.3 EXT_DIAG_OVERFLOW

This bit is set if more diagnostics data is present than will fit in the available diagnostics data area. For example, more channel diagnostics could be present than the send buffer or the receive buffer makes possible.

9.3 Diagnostics Processing from the System View

Inasmuch as it is bus-specific, the diagnostics information of the slaves is managed solely by the master interface (for example, IM308B).

All diagnostics from the application are made available to the S6 program via corresponding data bytes. If the **External Diagnostics bit** is set, the slaves to be diagnosed can already be evaluated in the diagnostics overview. Then, a special error routine can be called up, whereby the standard diagnostics information and the user-specific information can be evaluated.

After eliminating the current diagnostics situation, this can be signalled as a status message from the slave **without setting the external diagnostics bit**.

With the COM ET200, a comfortable diagnostics tool is available on-line. At the present time, identification-related diagnostics information can be displayed with it in plain text. In later phases, channel-related diagnostics will also be supported. User-specific diagnostics are only displayed if the EXT_DIAG bit is set.

The figure below shows a screen during data processing, for example:

Set Program File SINGLE DIAGNOSTICS	C:PNO4..ET.200	SIMATIC S5 / COM ET 200
Station Number: 30		Station Type: ET 200U-COMBI
Station Designation: Station Status: Device-Related Diagnostics	Station4 Slave not ready for data exchange External diagnostics Configuration error	
Identification-Related Diagnostics	KH = 01 Slot	
Active	3	
F1 F2 F3 F4 F5 F6 F7 F8 EXIT		

In the type file for the COM ET200 and in the GSD [device master data] file, fields are already provided for referencing device-specific bits and pertinent plain text messages (for example, Bit 7: „I have had it; good night!“).

10 Appendix B: Useful Information

10.1 Data format in the Siemens PLC SIMATIC

The SPC3 always sends data from the beginning of the buffer till the end. 16Bit values are shown in the Motorola format. For example:

Buffer pointer	high byte
Buffer pointer +1	low byte

10.2 Actual application hints for the DPS2 Software / SPC3

Please notice actual hints in our mailbox (++49 911-737972)

_____General _____

Static diagnosis

Problem:
A time-out of the DP-Buswatchdog forces the state-machine of the SPC 3 to fall back in state Wait_PRM with an appropriate influence of the diagnosis.
When the diagnosis is reconstructed, the "static diagnosis-bit" is set, which the Master recognizes during a restart of the bus-system.

Remedy:
After the sequence of the DP-Watchdog, a diagnosis update has to be performed.
This diagnosis update is already integrated in the standard software DPS 2 for the SPC 3.

Baudrate Search at 12 Mbaud

Problem:
When the SPC 3 is powered on, it is not able to find the baudrate sporadically, if the min.-slave-intervals are bigger than 2 ms. The master-modules send only one diag_req- and one gap-message for every min.-slave-interval. Otherwise there are just bus-messages received, which can't be used for the identification of the baudrate.

Remedy:
The min.-slave-interval has to be set less than 1.3 ms in the type-/GSD file, which is always possible at the SPC 3.

State Data_Exchange

Problem:
The SPC3 does'nt change to the DATA_EXCHANGE state until he gets the first inputs (Parameter and Configuration are ackknowledged positiv), like mentioned in the description.

Workaround:
The input data has to be updated during startup once.

Timing in the Asynchronous Mode

Problem:
At a certain constellation (for example: SAB 165 has a program-code in RAM with 0 wait-state access) access errors appear at the asynchronous interface (Motorola / Intel).
Necessary rest periods of the read / write signals have to be kept between the read / write cycles of the external memory and the following access to the SPC 3.

Workaround:
The SPC 3 specification has been updated with the appropriate data.
With a suitable programming of the bus-cycles, the rates can be maintained at the processors.

please refer the mailbox

_____Version V1.2_____
23.08.96

The version 1.2 of DPS2 for SPC3 contains the following improvements / supplements:

IM 182:

The IM 182 (PC-card with SPC3) is handled by the software package DPS2 with the compilers Microsoft C and Watcom C: The IM 182 can be addressed by adjustable interrupts or by polling. The MS compiler expands the standard makros faulty. Therefore certain makros had to be replaced with inline-functions.

IM 183:
The latest version of the KEIL-compiler (V5.x) works more exactly at the invertion of the bit-rates. Therefore "~" was replaced with "!" at certain locations.
_____Version V1.1_____
23.11.95

module dps2spc3.c

- In the function dps2_buf_init() the calculation of an list pointer is wrong. This may cause problems if a FDL data exchange is on the bus.

_____Version V1.0_____
14.11.95

module intspc3.c (example for a interupt module)

- Addition of the attribute SPC3_PTR_ATTR (= xdata) at *user_io_data_len_ptr
  => extern DPS2_IO_DATA_LEN SPC3_PTR_ATTR *user_io_data_len_ptr

09.11.95

module userspc3.c (example for a main module)

- delete RAM from 0x16H, not from 16d
- no initialization of the interrupt 1 level/egde

02.11.95

all modules

- the structure SPC3 can not be declared external in the headerfile spc3dps2.h. The locate instruction "_at_ address" in the main module would not operate.