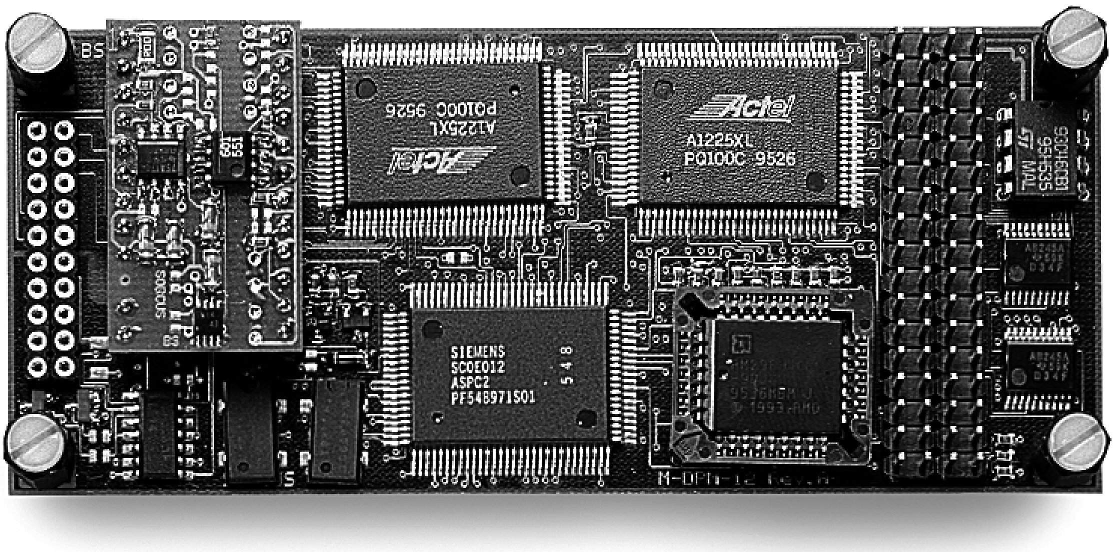
6. M-DPM-12

PROFIBUS DP Master up to 12 MBaud

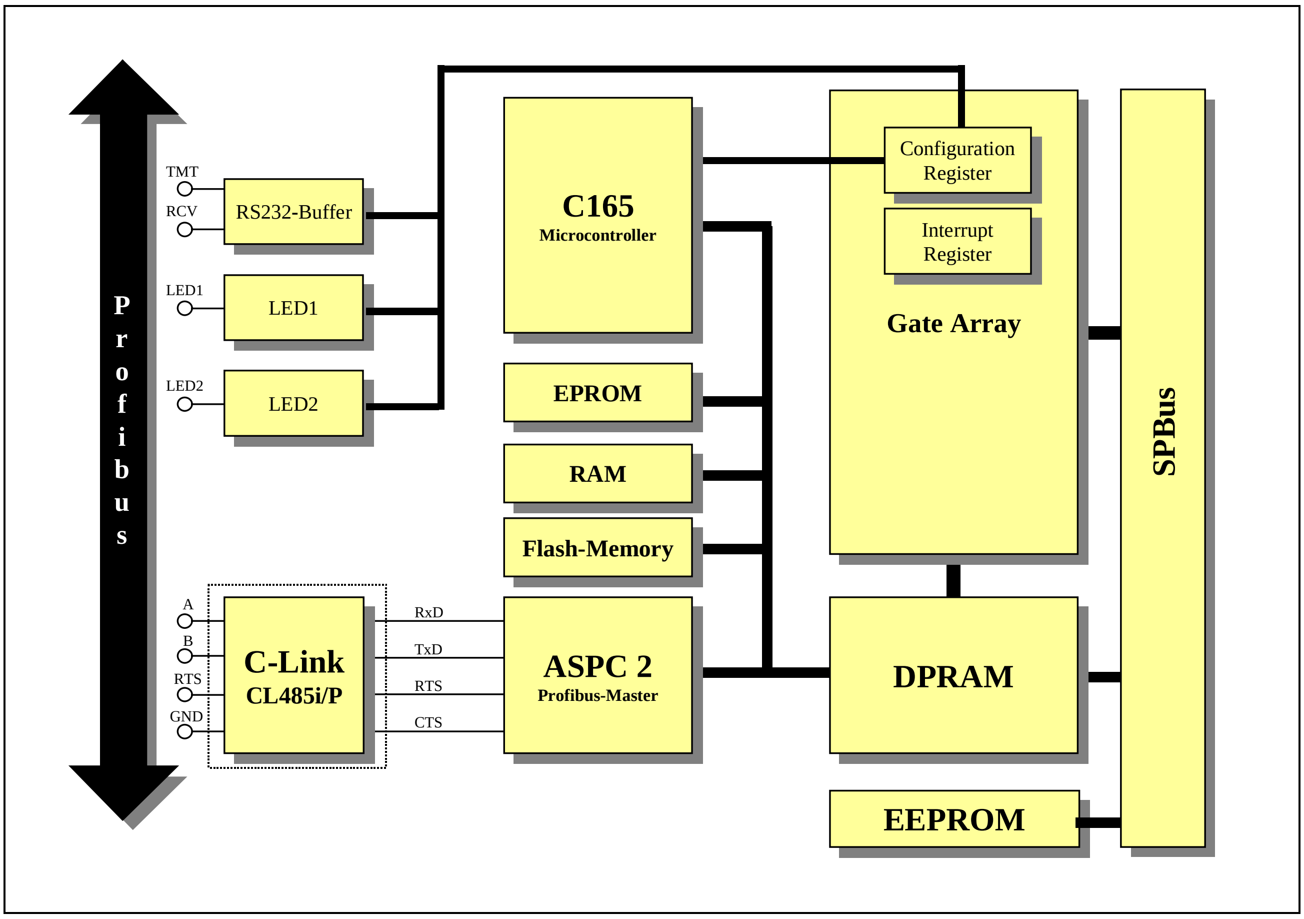


Functional Description

M-DPM-12 is an intelligent PROFIBUS-DP master module for SORCUS MODULAR-4/486 base boards. All baudrates are supported, including 12 MBaud. The interface used between the local intelligence (C165 microcontroller) and the MODULAR-4/486 is a dual-port RAM (DPRAM), over which commands and data are exchanged. The physical PROFIBUS interface is plugged on by a C-Link (Standard: RS-485, electrically isolated).

Before commissioning the module, you must insert the C-Link in the slot provided, since for operating the module the C-Link is absolutely essential (even if the module is not connected to the PROFIBUS). **!**

Block diagram



Technical data

Parameter	Value	Unit

PROFIBUS Controller		Siemens ASPC 2	-
Microcontroller 80C165		20	MHz
Dual-ported RAM		2 x 8	Kbyte
Static RAM		2 x 128	Kbyte
EPROM		512	Kbyte
Flash memory		512	Kbyte
Serial EEPROM for configuration data		32	Words
Interrupt capability to base board (interrupt channel software-selectable)		yes	-
Supply voltages[1] (from the base board)		+5, ±12	V
Power consumption			
(typical, externally nothing connected, LEDs 1 and 2 off, with C-Link CL485i/P):	+5 V	190	mA
	+12 V[1]	4	mA
	-12 V[1]	4	mA
Operating temperature range		0 to 60	°C
Dimensions (L x W x H)		106 x 45 x 15	mm

Items included in the delivery

The scope of delivery for this module comprises:

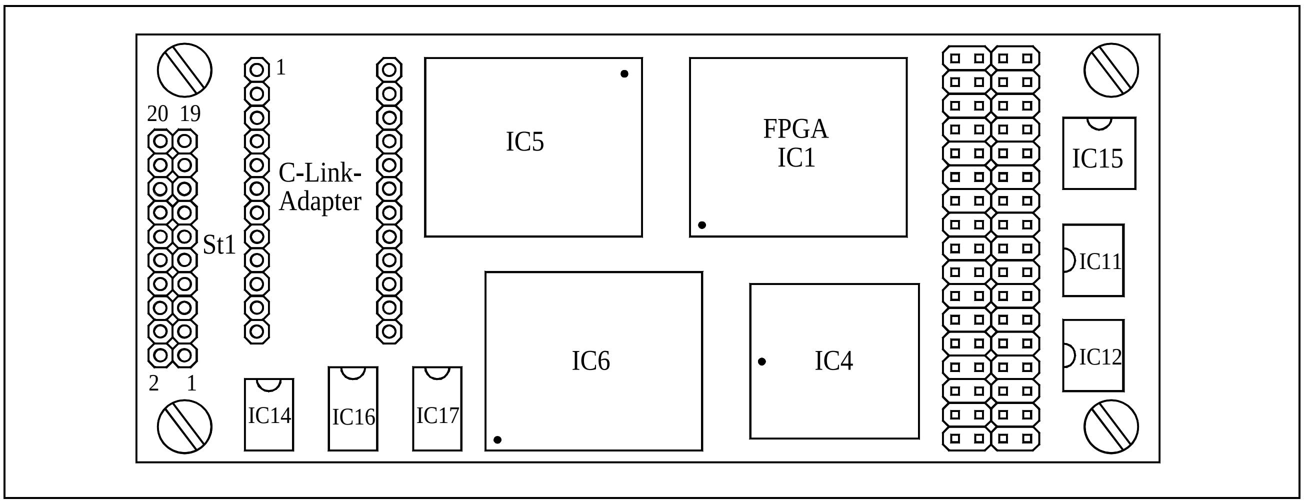
- the M-DPM-12 module itself
- 20-pole post plug connector for ribbon cable
- floppy disk with program libraries

[1] ±12 V are used only for the RS232 interface.

Configuration and installation

Before you plug the module onto the MODULAR-4/486, the C-Link adapter must be plugged on (Pin 1 is identified on the module and on the C-Link). The module and the C-Link do not contain any jumpers; all settings are made by software following installation.

Module Layout



EEPROM contents

A configuration in the EEPROM has already been pre-set in the factory:

WORD	Binary		Hex.	Significance (Brief info)
0	0010 0001	0011 0000	222ch	Module type M-DPM-12
1	0000 0000	0000 0000	0000h	Initialization
2	0000 0000	0000 0000	0000h	Interrupt channel to base board
3	0000 0000	0000 0000	0000h	Gate array configuration
4	0000 0000	0000 0000	0000h	LED setting
5	1001 0101	1111 0000	95f0h	Gate array configuration
6	0001 0101	1111 0000	15f0h	Gate array configuration
7	0000 0000	0000 0000	0000h	Timeout counter
8	0000 0000	0000 0000	0000h	Gate array version/revision (IC 2)
9	0000 0000	0000 0010	0002h	ASPC 2 version (IC 6)
10	0000 0000	0000 0000	0000h	Reserved
...	...	...	...	...
31	0000 0000	0000 0000	0000h	Reserved

! *The EEPROM contents of Words 2, 4 and 7 are used for saving an user-specific module configuration[1]. The EEPROM contents are not transferred directly (by hardware) into the appropriate registers of the module, but can be read by the user program, and utilized for programming the registers.*

[1] See section on "Programming".

WORD-0: Type and version of the module (must not be altered!)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	1	0	0	0	1	0	1	1	0	0	WORD-0: Identifier
▨	▨	▨	▨	▨	▨	▨	▨	0	0	1	0	1	1	0	0	Module type (2ch = 44 = M-DPM-12)
▨	▨	▨	▨	0	0	1	0	▨	▨	▨	▨	▨	▨	▨	▨	Revision: 1 = Rev. A, 2 = Rev. B
▨	▨	▨	0	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Reserved
0	0	1	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Identifier

WORD-1: Initialization

Only Bit 0 has a significance here at present. When this bit is set to 1, the module's registers will be configured and initialized after a hardware reset in accordance with the data in the EEPROM.

When Bit 0 = 0 has been set, the module will not be automatically configured and initialized after a hardware reset (or after system power-on).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORD-1: Initialization (factory setting)
																altered on: by:
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	Init after hard reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	▨	Reserved

WORD-2: Interrupt channel to base board

Here you can store the interrupt channel of the base board to which the module is connected.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORD-2: Interrupt channel (fact. setting)
																altered on: by:
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	Interrupt channel 0 = no interrupt 1 = IRQ-A 2 = IRQ-B 3 = IRQ-C 4 = IRQ-D 5 = IRQ-E 6 = IRQ-F 7 = NMI
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	▨	▨	▨	Reserved
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	▨	▨	▨	▨	XINTH interrupt enable 0 = Interrupt disabled 1 = Interrupt enabled
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	▨	▨	▨	▨	▨	Consistency conflict detection 0 = per timeout counter 1 = per interrupt
▨	▨	▨	▨	▨	▨	▨	▨	▨	0	▨	▨	▨	▨	▨	▨	Firmware timer interrupt 0 = Interrupt disabled 1 = Interrupt enabled
0	0	0	0	0	0	0	0	0	▨	▨	▨	▨	▨	▨	▨	Reserved

WORD-3: Gate array configuration (must not be altered!)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORD-5: Gate array configuration
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Gate array configuration

The gate array configuration has here been stored in memory **by the factory**.

WORD-4: LEDs 1 and 2

The status of the light-emitting diodes LED 1 and 2 after a reset can be stored here.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORD-4: LEDs 1 and 2 (factory setting)
																altered on: by:
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	LED 1 0 : off, 1 : on
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	▨	LED 2 0 : off, 1 : on
0	0	0	0	0	0	0	0	0	0	0	0	0	0	▨	▨	Reserved

WORD-5: Gate array configuration (must not be altered!)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	WORD-5: Gate array configuration
1	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	Gate array configuration

The gate array configuration has here been stored in memory **by the factory**.

WORD-6: Gate array configuration (must not be altered!)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	WORD-6: Gate array configuration
0	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	Gate array configuration

The gate array configuration has here been stored in memory **by the factory**.

WORD-7: Initialization value for the timeout counter

Here you can save the timeout counter value (in dependence on the PROFIBUS baudrate) for the consistency control.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORD-7: Timeout counter (factory setting)
																altered on: by:
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16 bit timer value

WORD-8: Gate array version (IC 2, must not be altered by the user)

Here the version of the gate array (IC 2) has been stored in memory **by the factory**.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	WORD-8: CPLD Version
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	0	Revision
▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	1	▨	▨	▨	▨	Version
0	0	0	0	0	0	0	0	▨	▨	▨	▨	▨	▨	▨	▨	Reserved

WORD-9: ASPC 2 Version (IC 6, must not be altered by the user)

Here the version of the ASPC 2 ASICs (IC 6) has been stored in memory **by the factory**.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	WORD-9: ASPC 2 Version
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	0	1	0	Step (0 = A, 1 = B, 2 = C, ...)
▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	0	▨	▨	▨	▨	Revision
0	0	0	0	0	0	0	0	▨	▨	▨	▨	▨	▨	▨	▨	Reserved

Plug connector assignments

Pin[1]	Signal	Significance	Pin	Signal	Significance
1	GND	Ground (PC)	11	DPPE	DPPE (optional)
2	+5 V	+5 Volt (PC), optional	12	DP5V	+ 5 Volt, isolated
3	RCV	C165 RS232	13	-	n.c.
4	LED1	LED 1	14	-	n.c.
5	TMT	C165 RS232	15	DPB	DPB, isolated
6	LED2	LED 2	16	DPA	DPA, isolated
7	-	n.c.	17	DPRTS	Request To Send, isol.
8	-	n.c.	18	-	n.c.
9	-	n.c.	19	DPGND	Ground, isolated
10	-	n.c.	20	-	n.c.

Table 6-1: Pin assignments for the post plug connector St1 (Rev. B)

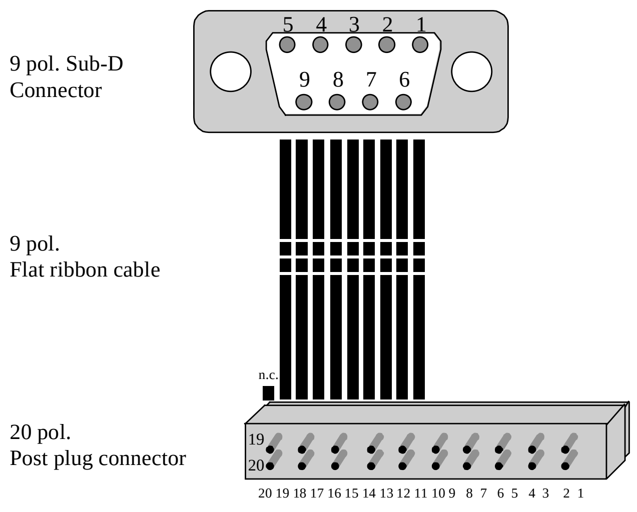


Fig. 6-1: Cable for M-DPM-12 at Siemens ET200 (e.g. SORCUS K2-2720)

[1] Pins 3 and 5 of the plug connector must not be connected!

Programming

This chapter is intended solely for those users who want to access the module's hardware directly (programming with I/O addresses).

The following function groups must be programmed on the module:

- Selection of an interrupt line from the module to the base board, setting the type of conflict detection / interrupt selection (see Interrupt Select Register ISR, 8 bits)
- Timeout counter for conflict handling (see Timeout-Register TOR, 16 bits)
- Setting LEDs 1 and 2 (see LED Register LER, 2 bits)
- The DPRAM as the interface to the module

After a hardware reset of the module (e.g. power-on or through the OsX operating system), all the module's writable registers and pointers = 0. The timeout counter is set to ffffh.

Initialization

Performing a module reset

There are two ways to reset the module. A write access operation to I/O address MBA + 07h (MBA = Module Base Address) executes a reset of the gate array and a reset of the PROFIBUS master. The PROFIBUS interface can also be reset separately (XRESET). The XRESET line is set using I/O address MBA + 05h. To initiate a reset, data bytes 8fh, 0fh and 8fh must be written one after the other onto the I/O address MBA + 05h. After the master has been reset, it will go into STOP status. Resetting the master takes approx. 1 second, and during this time the master must not be accessed.

When the gate array is reset, all internal registers are set to 0. The timeout counter is set to ffffh.

Downloading a binary data record

Parameterization of the PROFIBUS is handled by means of a binary data record, which can be created using the Siemens software COM ET 200. This binary data record must then be transferred onto the module once with the aid of the library (M044_LIB). A reset (see above) will then be required in order to activate the new parameters[1].

LEDs 1 and 2

The two LEDs on the module can be programmed separately or together. A write access operation to the LED register LER sets the status of the LEDs (0 = off, 1 = on).

Note that only Bits 0 to 3 are used; Bits 4 to 7 are invalid, and should be set to 0.

Bit 0	**LED 1 selection**
0	LED 1 unchanged
1	LED 1 is set as per Bit 2

Bit 1	**LED 2 selection**
0	LED 2 unchanged
1	LED 2 is set as per Bit 3

Bit 2	**LED 1 output value**
0	LED 1 off
1	LED 1 on

Bit 3	**LED 2 output value**
0	LED 2 off
1	LED 2 on

The status of the light-emitting diodes and the LED register (LER) can also be read back from the base board (a total of 2 bits, since Bits 2 to 7 are invalid).

[1] See section on "Commissioning".

Selecting an interrupt line from module to base board

The module has an interrupt capability, i.e. it can trigger an interrupt to the base board in response to certain events (pos. edge). The module’s interrupt line can be software-connected to one of the base board’s interrupt inputs. An interrupt is selected by setting the module’s interrupt select register ISR, Bits 0 to 2.

There are three independent interrupt sources:

1. Through a request by the master (ASPC 2/C165) over the XINTH line: this interrupt can, for example, be used to call the error handling function in response to a system error. This interrupt source is activated by a software reset (see the section entitled "Standard-language library").
2. The consistency control can trigger an interrupt in response to a consistency conflict (see the section entitled "Standard-language library").
3. In addition, the firmware timer can likewise be used for triggering an interrupt via the XTESTO line.

Bits 4 to 6 of the ISR are used to select the interrupt source. Bit 4 selects interrupts from the DPRAM via XINTH. Bit 5 can be used to specify whether the consistency status bit can trigger an interrupt or whether the internal timeout counter is used (no interrupt). Bit 6 can be used to set whether the firmware timer can trigger interrupts. A set bit (=1) activates the interrupt source concerned.

While the interrupt channel is being set, the module must not request an interrupt, and on the base board the interrupts must be (temporarily) masked. Bit 3 and Bit 7 of the ISR are reserved, and should be set to 0.

Selecting the interrupt line to the base board:

Bit 2	Bit 1	Bit 0	Interrupt line of the MODULAR-4/486
0	0	0	none
0	0	1	IRQ-A
0	1	0	IRQ-B
0	1	1	IRQ-C
1	0	0	IRQ-D
1	0	1	IRQ-E
1	0	1	IRQ-F
1	1	1	NMI

Possible interrupt sources:

Bit 4	Master Interrupt Enable (XINTH)[1]
0	No interrupt
1	Interrupt when XINTH = 0

Bit 5	Consistency conflict control/detection
0	Consistency request by ASPC 2 triggers consistency timer
1	Interrupt in response to consistency request by ASPC 2

Bit 6	Firmware Timer Interrupt Enable
0	No interrupt
1	Periodic interrupt through firmware timer

The Interrupt Select Register (ISR) can also be read by the base board. Note that Bits 3 and 7 are reserved/invalid.

[1] See also section on "Programming".

Deleting an interrupt request:

An interrupt request can be read by the user in the Interrupt Status Register (IST). The master interrupt request is maintained (XINTH = 0) until the user performs an interrupt acknowledge (by means of a read access operation to DPRAM address 1ffeh). The consistency status bit is deleted by setting or deleting a consistency request. A firmware timer interrupt must be deleted by a write access operation on the Interrupt Select Register (ISR).

Accessing the dual-port RAM (DPRAM)

The DPRAM on the module is accessed using a programmable address pointer (14 bits). To enable the dual-ported RAM to be accessed both bytewise and wordwise; Bit 0 of the address pointer selects whether the low or the high byte is used for byte access operations. Irrespective of this, when the DPRAM is accessed via the I/O address it is specified whether the read or write operation will be performed wordwise or bytewise. In the case of word accessing, the system accesses the word currently selected, irrespective of Bit 0.

Since the I/O access operations of the base board onto the DPRAM are always performed wordwise, the user software must swap the data appropriately at need in the case of high-byte access operations.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DPRAM address pointer
0	0	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	14 bit address pointer (A0 to A13) A0 selects low or high byte for byte access operations (0 = low, 1 = high)
▨ 0	▨ 0															Reserved

When the DPRAM is accessed, the I/O address can be used to select whether the address pointer is to be automatically incremented after completion of the access operation concerned. In the case of word accessing, the address pointer will then be incremented by 2, and in the case of byte accessing by 1. The valid address range is from 0000h to 3fffh.

Addressing the dual-ported RAM:

A0 = 0, Low-Byte	A0 = 1, High-Byte
3ffeh	3fffh
.	.
.	.
0000	0001

To enable the DPRAM to be accessed, the following operations are required:

First of all, the DPRAM address pointer must be set to the address earmarked for the read or write operation, by means of an I/O write access operation (MBA + 00h). The subsequent read or write access operations onto the DPRAM are then executed using this address pointer: performing an 8- or 16-bit read or write access operation, where appropriate with auto-incrementation of the address pointer by 1 or 2.

Consistent access operations

Consistent access operations onto the DPRAM are initiated by setting the read or write consistency line (consistency request). This must be followed immediately by actual access to the DPRAM. When all access operations have been carried out, the consistency request must be cancelled. This is done by performing a read access operation on the module status register. When both the base board and the module want to access the DPRAM consistently at the same time, a conflict control function is activated. Note that the ASPC 2 has priority over the host, i.e. after a defined time period (timeout) the host has to cancel its consistency request.

The module’s conflict control function regulates the sequence of access operations so as to ensure that the host (i.e. the base board) either cannot perform any valid access operations after expiry of an internal timeout counter, or an interrupt is triggered as soon as the ASPC 2 wants to implement consistent access. In this case, the host still has a certain amount of time (see below, depends on the PROFIBUS baudrate) available for finishing its access operations.

Conflict control

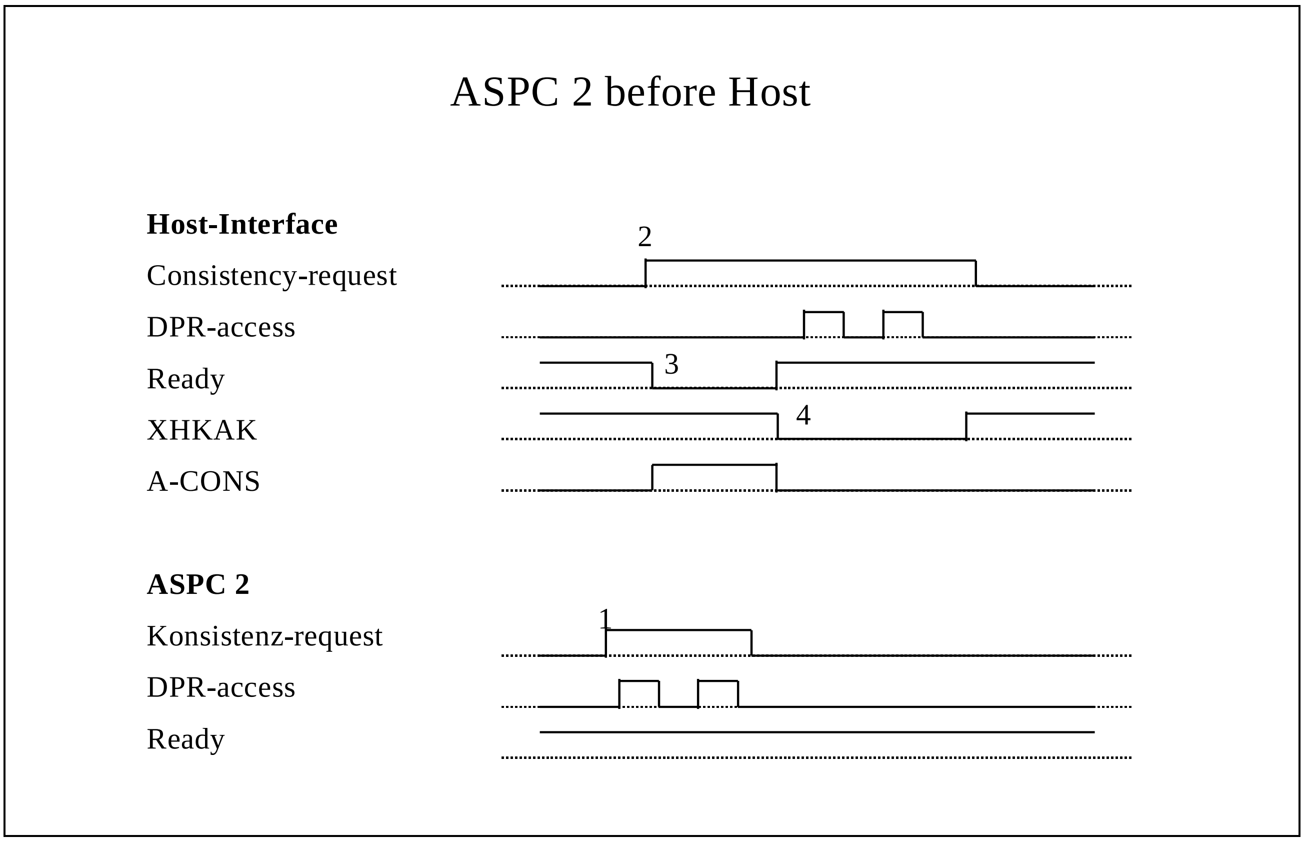
A few brief words of explanation to start with on the control lines used:

XRHCONS	Consistency read request (low-active) from the base board
XWHCONS	Consistency write request (low-active) from the base board
XHKAK	Consistency acknowledge (confirmation, low-active) from the ASPC 2, consistent access operations from the base board enabled
A-CONS	Consistency request (high-active) from the ASPC 2; the base board can access only when A-CONS = 0 and XHKAK = 0
RDYH	Ready line to the base board (high-active) Access operations from the base board can be delayed in the event of a conflict
RDYP	Ready line to the ASPC 2 (high-active) Access operations from the ASPC 2 can be delayed in the event of a conflict

Base board accesses the ASPC 2

When the ASPC 2 is already consistently accessing, and the base board also wishes to access, the base board's access operation will be automatically braked (*RDYH-withdrawal*). As soon as the ASPC 2 has finished its access operations, the base board is enabled for accessing, and can execute its access operations.

The user can also, after he/she has set the consistency request, interrogate the A-CONS line. As soon as A-CONS = 0 (i.e., the ASPC 2 is no longer consistently accessing), access operations can be performed to the DPRAM.



Sequence:

1. The ASPC 2 requests consistent access operations.
2. The host requests consistent access operations (somewhat later).
3. The host accesses without waiting for XHKAK, and is braked by means of "ready" withdrawal.
4. The ASPC 2 finishes its access operation so that the host can begin.

ASPC 2 accesses the base board

As long as valid base board access operations are ongoing, the *XHKAK* line (consistency *a*cknowledge) is activated.

If during a consistent access (consistency request of the MODULAR-4/486 board active) the ASPC 2 likewise wants to execute consistent accessing, it will be automatically braked (*RDYP withdrawal*). However, the base board’s consistency request must have been finished 80 µs at the latest (with 12 MBaud) after the request of the ASPC 2, since the latter will otherwise not be able to comply with the cycle time.

Conflict control with timeout counter

The gate array starts a timer as soon as the ASPC 2 wants to execute consistent accessing (*ACONS* is activated and *XHKAK* is active). If the timer exceeds the preset timeout time, the base board's access operation is aborted (XCSHOST = 1, consistency request terminated). This means the ASPC 2 can now begin its own access operations.

You must, after your last consistent access operation, terminate the consistency request. You do this by a read access operation, which additionally returns the status of the TIMEOUT-STATUS, R-CONS, W-CONS, A-CONS, XHKAK, XCSDPR2 lines and the CONSISTENCY STATUS BIT. If TIMEOUT-STATUS = 1, this means the timeout counter has expired, and thus the access operation or the data from the host are not valid; the entire access operation must then be repeated. And the status must be erased beforehand.

$$t_{TIMEOUT} = \frac{8 \cdot \mathrm{NoOfByte}_{FIFO}}{\mathrm{Baudrate}_{PROFIBUS}}$$

The timeout counter can be set from 0 to 0.65 s. This is done using the Timeout Counter Register TOR (16 bits). The timeout time depends on the PROFIBUS baudrate used, and the FIFO size of the ASPC 2.

In the case of an ASPC 2 Step C (128 Byte FIFO), the following max. timeout times are obtained:

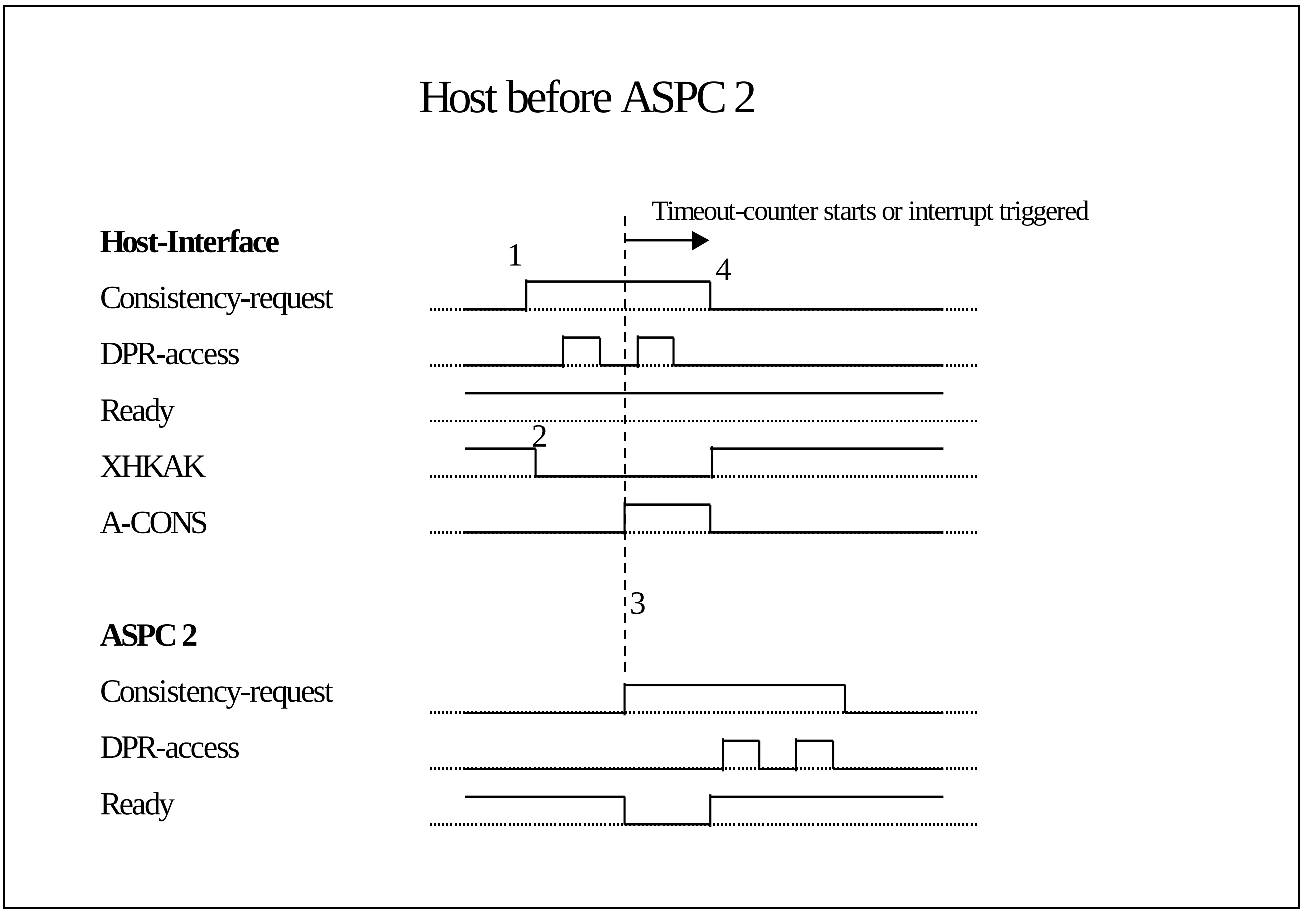
Possible timeout values are, for example:	84 µs at 12 MBaud
	to
	100 ms at 9600 Baud

In the case of an ASPC 2 Step B (64 Byte FIFO), the timeout times concerned are halved!

The 16 bit timer value (TOR) can be set by means of a write access operation to I/O address MBA + 08h.

Conflict control with interrupt

If during consistent accessing by the host a consistency request is made by the ASPC 2 (A-CONS = 1, see above), an interrupt to the base board can be triggered (CONSISTENCY STATUS BIT = 1). The host then has to ensure that the access operations are terminated in good time. The timeout counter is not used here. If the consistency request is not terminated in good time, or if the access operation overruns the timeout time, this may lead to an error state of the PROFIBUS.



Sequence:

1. The host requests consistent accessing.
2. XHKAK enables the access operations, since ASPC 2 is not requesting consistency.
3. ASPC 2 requests consistency, but is braked by "ready" withdrawal. The timeout counter is started, or an interrupt to the base board triggered. The host consistency must be terminated before the timer expires or the timeout time has elapsed.

4. Host consistency terminated; ASPC 2 begins accessing.

General sequence

If consistent accessing is to be executed to the DPRAM, the sequence looks like this:

1. **Set the read or write consistency request** (MBA + 18h, Data = 01h for read, Data = 02h for write).
2. **Read module status** (MBA + 19h) until XHKAK = 0 (Bit 5, consistency acknowledge).
3. **Set the DPRAM pointer** to the address for reading or writing.
4. **Execute 8 or 16-bit read or write access**, if required with auto-incrementation of the address pointer.
5. **Repeat points 3 (optional) and 4** until all data have been transferred.
6. **Erase the consistency request** by reading MBA + 18h. Note that the module status is outputted automatically. If CONSISTENCY-STATUS = 1, then a consistency request the ASPC 2 has occurred during consistent host accessing. Only when the TIMEOUT-STATUS = 1 too will there be an actual error. Then the entire access operation must be repeated by the host; all data are invalid. In addition, the status must be erased by a write access operation to MBA + 18h with Data = 0, to make access operations to the DPRAM possible again.

High-Level Language Libraries

For information on how the library is integrated and used, please consult the "High-Level Language Libraries" section in the introduction. The name of the library (*libname*) is **M044_LIB**, and you will find it in the (*pathname*) directory **MODULE**. Before all other routines, the **m044_bib_startup** procedure must be called once.

Using the library with the M044TASK driver task

Together with the M044_LIB libraries, the package supplied also includes an NI-task M044TASK.EXE and an associated task library M044_MDD. The M044_MDD library has the same functions and procedures as the M044_LIB, but all PROFIBUS functions are executed using a task function. The advantage of this is that in a multi-tasking environment several applications can share one module. Any conflicts arising are intercepted by the task. The M044_MDD library is suitable both for PC programs and realtime programs. You can use the header files of the M044_LIB to integrate the library.

The M044TASK.EXE task has the program number 329h, and must be installed on the MODULAR-4/486 with the flags 0180h. A global variable *tasknum* of the *ushort* type must be declared *external* by the application program, and have been preset with the task number. This can also be done with m044_set_tasknum (task number) before calling the first library function.

Error handling

All library functions supply an error status as a return value, which has to be evaluated by the application program.

Error 1bh	No M-DPM-12 on the specified module slot, or the slot specified in the function does not coincide with the slot set using m044_configure.
Error 40h	Error in executing the function; the error diagnosis must be read from the error buffer with "get_error".
Error 41h	Function cannot be executed at present, since occupied by another application.

After you have called a library function, if there is an error the diagnostic message must be read and appropriately evaluated. Function (1) is used for this purpose.

m044_get_error Read error diagnosis

Pascal	FUNCTION m044_get_error (micro_slot: byte, VAR data) : word
C	ushort EXPORT m044_get_error (byte micro_slot, ushort *data);
Function	The error diagnosis is read.
Parameter	*data*: The 1st word contains the Function Number (x = 1..33) at which the error has occurred, with the diagnostic data provided as from Word 2 (currently always one word). A total of 10 words are read.

Note: If a library function supplies a timeout error in the diagnosis, this means a command to the master has not been acknowledged. One reason for this may be that the C-Link has not been plugged on.

Configuration

m044_bib_startup — Initialize module library

Pascal	FUNCTION m044_bib_startup : word;
C	ushort EXPORT m044_bib_startup (void);
Function	This function (2) initializes the module library. The initialization data, for instance, are transferred into the library from the EEPROMs of all M-DPM-12 modules plugged on.

m044_configure — Configure module

Pascal	FUNCTION m044_configure (micro_slot: byte) : word;
C	ushort EXPORT m044_configure (byte micro_slot);
Function	This function (3) configures a module in accordance with the specifications in the module's EEPROM.

m044_get_master_mode — Read current master mode

Pascal	FUNCTION m044_get_master_mode (micro_slot; VAR status: byte): word;
C	ushort EXPORT m044_get_master_mode (byte micro_slot, byte *status);
Function	This function (4) reads the current master mode for coordination when the master is run up. If after a hardware reset (e.g. power-on) the master mode is _M044_STOP, this means the module is ready for operation.

Parameter	*status*:	_M044_STOP:	Master is in STOP mode
		_M044_CLEAR:	Master is in CLEAR mode
		_M044_OPERATE:	Master is in OPERATE mode

Parameterizing the PROFIBUS

Binary data for PROFIBUS parameterization

For PROFIBUS parameterization, the module requires what is called a binary data record. This binary data record can be created with the PC program COM ET 200 (Siemens) under Windows 3.1, and saved in a binary file. This binary file has to be transferred to the module (download).

Binary data can be transferred both from the host to the module (download) and from the module to the host (upload). The master mode must be "STOP".

m044_download — Transfer binary data

Pascal: FUNCTION m044_download (micro_slot: byte; VAR data; data_size: byte; flag: byte) : word;

C: ushort EXPORT m044_download (byte micro_slot, void *data, byte data_size, byte flag)

Function: This function (5) transfers binary data from the host to the module. A binary file (size: approx. 7 KByte) has to be downloaded onto the module in several steps, since only data blocks of max. 200 bytes can be transferred at a time. No other library functions may be called while the blocks are being transferred.

Parameter	*data*:	Pointer to the data block for transfer
	data_size:	Size of the data block (0 to 200) in bytes
	flag:	_M044_NO_MORE_DATA: no further binary data _M044_MORE_DATA: more binary data to follow
Diagnosis	*0*:	Command correctly executed
	2:	Error: timeout, function aborted
	256:	Error: command semaphore not set

Note: When all data blocks have been transferred, the module has to be reset (with the **m044_reset_master(..)** function). The master must then be in the **"STOP"** operating status. If this is not the case, this means that either binary file transmission or the binary file itself was faulty.

From a PC program, the binary data available as a binary-data file can be transferred from the host to the module with the following function (6). The module then has to be reset.

m044_download_file — Transfer binary file

Pascal	FUNCTION m044_download_file (micro_slot: byte; binary_file : str80; VAR downloaded : word) : word;
C	ushort EXPORT m044_download_file (byte micro_slot, const void *binary_file, ushort *downloaded)
Function	Transferring a binary file to the module.
Parameter	*binary_file*: Name of the binary file
	downloaded: Number of bytes transferred
Diagnosis	*0*: Command correctly executed
	1: Error when reading file
	256: Error: command semaphore not set

m044_upload — Read binary data

Pascal	FUNCTION m044_upload (micro_slot: byte; VAR data; VAR data_size: byte; VAR flag: byte) : word;
C	ushort EXPORT m044_upload (byte micro_slot, void *data, byte *data_size, byte *flag)
Function	This function (7) transfers binary data from the module to the host. A binary file will usually be about 7 KByte in size. Since only 250 bytes can be transferred, it is necessary to call this function several times. Before the first call, _M044_START_DATA must be entered in the *flag* parameter.

Parameter	*data*:	Pointer on destination data area
	data_size:	Number of binary data transferred (in bytes)
	flag:	transfer parameter: _M044_START_DATA: 1st block for transfer _M044_NO_MORE_DATA: all binary data transferred _M044_MORE_DATA: Transfer incomplete
Diagnosis	*0*:	Command correctly executed
	2:	Error: timeout, function aborted
	5:	Error: incorrect command sequence
	256:	Error: command semaphore not set

From a PC program, the following function (8) can be used to transfer the binary data available from the module to the host, and save them in a file:

m044_upload_file — Read and save binary file

Pascal FUNCTION m044_upload_file (micro_slot: byte; binary_file : str80; VAR uploaded : word) : word;

C ushort EXPORT m044_upload_file (byte micro_slot, const void *binary_file, ushort *uploaded)

Parameter	*binary_file*:	Name of binary file
	uploaded:	Number of bytes transferred
Diagnosis	*0*:	Command correctly executed
	1:	Error in writing the file
	5:	Error: incorrect command sequence
	256:	Error: command semaphore not set

m044_set_slave_address Set slave address

Pascal	FUNCTION m044_set_slave_address (micro_slot: byte; VAR data; size: byte) : word;	
C	ushort EXPORT m044_set_slave_address (byte micro_slot, void *data, byte size);	
Function	This function (9) sets a slave's address. This is conditional on it being possible to program the slave over the bus. The function is required for each slave whose address is greater than 123 due to design constraints. Slaves newly addressed must already have been entered with their correct addresses during planning work with COM ET 200.	
Parameter	*data*:	Pointer on slave parameter structure 1st byte: new slave address 2nd byte: address change identifier 3rd byte: old or current slave address Optionally, you can transfer additional slave data as per standard (see manual for slave).
	size:	Size of slave parameter structure (3 to 250 bytes, see above)
Diagnosis	*0*:	Command correctly executed
	1:	Error: slave cannot process service
	2:	Error: slave has no resources
	3:	Error: command not activated at slave
	5:	Error: bus short-circuit
	6:	Error: timeout, function aborted
	159:	Error: slave does not answer
	175:	Error: slave answers incorrectly
	191:	Error: bus malfunction
	256:	Error: command semaphore not set

m044_restart — Reset master software

Pascal	FUNCTION m044_restart (micro_slot: byte; restart_par: byte) : word;
C	ushort EXPORT m044_restart (byte micro_slot, byte restart_par);
Function	This function (10) resets the master software on the module. This software only needs to be reset after a download or the setting of a slave address, so as to activate the new PROFIBUS configuration.

Parameter *restart_par*: Operating mode to be assumed after restart:

_M044_STOP:	Master in STOP mode
_M044_CLEAR:	Master in CLEAR mode
_M044_OPERATE:	Master in OPERATE mode

This is used to adopt the bus parameters currently set (e.g. those after a download). If you want to adopt different bus parameters, you can link one of the following parameters to the operating mode parameter (=bitwise OR):

_M044_DEFAULT_PAR:	Default parameter
_M044_EPROM_PAR:	Parameter from EPROM

If an interrupt to the base board is to be triggered on the module if a PROFIBUS system error occurs, then another OR conjunction is required, with the following parameter:

_M044_ENABLE_IRQ:	System error interrupt

Diagnosis

0:	Command correctly executed
1:	Error: timeout, function aborted
256:	Error: command semaphore not set

m044_refresh_slave_datastruct — Update data structure buffer

Pascal	FUNCTION m044_refresh_slave_datastruct (micro_slot: byte):word;
C	ushort EXPORT m044_refresh_slave_datastruct (byte micro_slot);
Function	This function (11) transfers the data structure information of all parameterized slaves into the data structure buffer of the module library. This function has to be called after a restart.

Diagnosis	*0*:	Command correctly executed
	1:	Error: timeout, function aborted
	5:	Error: incorrect command sequence
	256:	Error: command semaphore not set

Master control

m044_set_master_mode — Set master operating mode

Pascal	FUNCTION m044_set_master_mode (micro_slot: byte;
	command: byte) : word;

C	ushort EXPORT m044_set_master_mode (byte micro_slot, byte command);
Function	This function (12) controls the master’s operating mode.

Parameter	*command*:	_M044_CLEAR:	Master in CLEAR mode
		_M044_STOP:	Master in STOP mode
		_M044_OPERATE:	Master in OPERATE mode
		_M044_SYNCHRONIZE:	Synchronization command between host and master

Diagnosis	*0*:	Command correctly executed
	1:	Error: timeout, function aborted
	256:	Error: command semaphore not set

m044_watchdog_enable — Activate master's watchdog

Pascal	FUNCTION m044_watchdog_enable (micro_slot: byte; timeout: word) : word;
C	ushort EXPORT m044_watchdog_enable (byte micro_slot, ushort timeout);
Function	This function (13) activates or de-activates the master's watchdog. When the watchdog is activated, the user software has to make sure that the watchdog is reset within the preset time interval.

Parameter	*timeout*:	Timeout as a factor of 10 ms (0 to 65535) *timeout* = 0 de-activates the watchdog
Diagnosis	*0*:	Command correctly executed
	1:	Error: timeout, function aborted
	256:	Error: command semaphore not set

m044_watchdog_retrigger — Trigger the watchdog

Pascal	FUNCTION m044_watchdog_retrigger (micro_slot: byte) : word;
C	ushort EXPORT m044_watchdog_retrigger (byte micro_slot);
Function	This function (14) retriggers the module's watchdog, and has to be called cyclically by the user program. If the watchdog is expired, the master will halt, and no longer access the slaves, whereupon the slaves can go to a defined status (with response monitoring switched on). In order to re-activate the master, the module or the master must be reset!

Diagnosis	*0*:	Command correctly executed
	>0:	Error: timeout at retriggering

Resetting the module

m044_reset_master — Reset the master

Pascal	FUNCTION m044_reset_master (micro_slot: byte) : word;
C	ushort EXPORT m044_reset_master (byte micro_slot);
Function	This function (15) executes a reset of the master. The register contents of the module remain unchanged. After the master has been reset, it will be in "STOP" status. Resetting the master takes several milliseconds. During this time, the master must not be accessed!

m044_hard_reset — Reset module

Pascal	FUNCTION m044_hard_reset (micro_slot: byte) : word;
C	ushort EXPORT m044_hard_reset (byte micro_slot);
Function	This function (16) resets the entire module (gate array and PROFIBUS master). After the master has been reset, it will be in "STOP" status. Resetting the master takes several milliseconds. During this time, the master must not be accessed!

Slave accessing

Defining the data channels

The link between the PROFIBUS master and the PROFIBUS slaves is designated below as a data input channel or a data output channel. To enable a slave’s inputs and outputs to be selectively accessed, the data channels are divided up into sub-channels. These sub-channels are numbered upwards, beginning with 0, according to the scheme below (example for a 16-byte data channel):

Relative offset	**Byte accessing**	**Word accessing**
0	Sub-channel 0	Sub-channel 0
1	Sub-channel 1	
2	Sub-channel 2	Sub-channel 1
3	Sub-channel 3	
:	:	:
14	Sub-channel 14	Sub-channel 7
15	Sub-channel 15	

m044_set_slave_byte
m044_set_slave_word — Write byte or word

Pascal	FUNCTION m044_set_slave_byte (micro_slot: byte; slave: byte; subchannel: byte; data: byte; cons: byte) : word;
Pascal	FUNCTION m044_set_slave_word (micro_slot: byte; slave: byte; subchannel: byte; data: word; cons: byte) : word;
C	ushort EXPORT m044_set_slave_byte (byte micro_slot, byte slave, byte subchannel, byte data, byte cons);
C	ushort EXPORT m044_set_slave_word (byte micro_slot, byte slave, byte subchannel, ushort data, byte cons);
Function	These functions (17 and 18) write an individual datum (byte or word) into a slave's output sub-channel.

Parameter	*slave*:	Slave address (3 to 123)
	subchannel:	Sub-channel
	data:	Datum to be set, of the byte or word type
	cons:	Consistency flag: _M044_NO_CONS: without consistency _M044_CONS: with write consistency
Diagnosis	*0*:	Command correctly executed
	100:	Error: bus differently parameterized
	101:	Error: consistency conflict (write operation must be repeated)

m044_set_slave_block — Write data block

Pascal	FUNCTION m044_set_slave_block (micro_slot: byte; slave: byte; word : offset; size: byte; VAR data; cons:byte) : word;
C	ushort EXPORT m044_set_slave_block (byte micro_slot, byte slave, ushort offset, byte size, void *data, byte cons);
Function	This function (19) writes a data block, beginning with Sub-channel 0 + offset into a slave's data input channel.

Parameter	*slave*:	Slave address (3 to 123)
	offset:	Offset on Sub-channel 0
	size:	Number of bytes for setting
	data:	Pointer on source data buffer (block for setting)
	cons:	Consistency flag: _M044_NO_CONS: without consistency _M044_CONS: with write consistency
Diagnose	*0*:	Command correctly executed
	100:	Error: bus differently parameterized
	101:	Error: consistency conflict (write operation must be repeated)
	102:	Error: size > slave's buffer size

m044_get_slave_byte
m044_get_slave_word **Read byte or word**

Pascal	FUNCTION m044_get_slave_byte (micro_slot: byte; slave: byte; subchannel: byte; VAR data: byte; cons: byte) : word;
Pascal	FUNCTION m044_get_slave_word (micro_slot: byte; slave: byte; subchannel: byte; VAR data: word; cons: byte) : word;
C	ushort EXPORT m044_get_slave_byte (byte micro_slot, byte slave, byte subchannel, byte *data, byte cons);
C	ushort EXPORT m044_get_slave_word (byte micro_slot, byte slave, byte subchannel, ushort *data, byte cons);
Function	These functions (20 and 21) read an individual datum (byte or word) from a slave's input sub-channel.

Parameter	*slave*:	Slave address (3 to 123)
	subchannel:	Sub-channel
	data:	Pointer on destination data buffer (byte or word)
	cons:	Consistency flag: _M044_NO_CONS: without consistency _M044_CONS: with read consistency
Diagnosis	*0*:	Command correctly executed
	100:	Error: bus differently parameterized
	101:	Error: consistency conflict (read operation must be repeated)

m044_get_slave_block — Read data block

Pascal	FUNCTION m044_get_slave_block (micro_slot: byte; slave: byte; offset : word, size: byte; VAR data; cons: byte) : word;		
C	ushort EXPORT m044_get_slave_block (byte micro_slot, byte slave, ushort offset, byte size, void *data, byte cons);		
Function	This function (22) reads a data block, beginning with Sub-channel 0 + offset from a slave's data input channel.		
Parameter	*slave*:	Slave address (3 to 123)	
	offset:	Offset on Sub-channel 0	
	size:	Number of bytes for reading	
	data:	Pointer on destination data buffer	
	cons:	Consistency flag:	
		_M044_NO_CONS:	without consistency
		_M044_CONS:	with write consistency
Diagnosis	*0*:	Command correctly executed	
	100:	Error: bus differently parameterized	
	101:	Error: consistency conflict (read operation must be repeated)	
	102:	Error: size > slave's buffer size	

m044_set_slave_command — Give order to slave or slave group

Pascal	FUNCTION m044_set_slave_order (micro_slot: byte; order: byte; group: byte; slave: byte) : word;
C	ushort EXPORT m044_set_slave_order (byte micro_slot, byte order, byte group, byte slave);
Function	This function (23) gives an order to a slave or a slave group. Groups can be defined using COM ET 200.

Parameter	*order*:	_M044_FREEZE:	Freeze inputs
		_M044_UNFREEZE:	Terminate freezing
		_M044_SYNC:	Set outputs to "synchronous"
		_M044_UNSYNC	Set outputs to "asynchronous"
	group:	Group Number (1..255, for 0: enter 255)	
	slave:	Slave address (3..123) or _M044_BROADCAST	
Diagnosis	*0*:	Command correctly executed	
	1:	Error: timeout, function aborted	
	2:	Error: command not permitted	
	3:	Error: command not permitted to slave or slave group	
	256:	Error: command semaphore not set	

m044_get_slave_datastruct — Read a slave's data structure

Pascal	FUNCTION m044_get_slave_datastruct (micro_slot: byte; slave: byte; VAR data: m044_slv_datastr_type) : word;
C	ushort EXPORT m044_get_slave_datastruct (byte micro_slot, byte slave, m044_slv_datastr_type *data);
Function	This function (24) reads a data structure from the library's data structure buffer. It is assumed that the data structure buffer contains the current data structures of all parameterized slaves (see m044_refresh_slave_datastruct).

Parameter	*slave*:	Slave address (3..123)
	data:	Address of a data structure variable of the type m044_slv_ datastr_type (see table below)
Diagnosis	*0*:	Command correctly executed
	1:	Error: slave number not valid

The variable type m044_slv_datastr_type is a data structure consisting of the following fields:

Field	**Data type**	**Significance**
.inp_ptr	ushort	DPRAM address of the input data
.outp_ptr	ushort	DPRAM address of the output data
.diag_ptr	ushort	DPRAM address of the diagnostic data
.diag_len_ptr	ushort	DPRAM address of the diagnosis length
.diag_cnt_ptr	ushort	DPRAM address of the diagnosis counter
.inp_len	byte	Number of input data bytes
.outp_len	byte	Number of output data bytes
.inout	byte	Information on the consistency of the inputs/outputs
.slave_type	byte	Type of slave

m044_get_slave_diagnosis_list Read diagnosis list of all slaves

Pascal	FUNCTION m044_get_slave_diagnosis_list (micro_slot: byte; VAR data: m044_slv_bitmap_type) : word;
C	ushort EXPORT m044_get_slave_diagnosis_list (byte micro_slot, m044_slv_bitmap_type *data);
Function	This function (25) reads the slave diagnosis list. Each bit of the data read represents one slave (example: Slave 17=Bit-1 of the field slave_16_31). A set bit means that the slave involved has reported a diagnosis.
Parameter	*data*: Address of a structure variable of the type m044_slv_ bitmap_type (see table below)

The variable type m044_slv_bitmap_type is a data structure consisting of the following fields:

Field	**Data type**	**Significance**
.slave_0_15	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_16_31	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_32_47	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_48_63	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_64_79	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_80_95	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_96_111	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_112_123	ushort	Bit field (Bit-0 .. Bit-11 valid)

m044_check_slave_diagnosis — Check whether slave has reported diagnosis

Pascal	FUNCTION m044_check_slave_diagnosis (micro_slot; slave; VAR diaglen: byte) : word;
C	ushort EXPORT m044_check_slave_diagnosis (byte micro_slot, byte slave, byte *diaglen);
Function	This function (26) checks whether the slave specified has reported a diagnosis, and where appropriate will return the number of diagnostic data in "’diaglen".

Parameter	*slave*:	Slave address (3..123)
	diaglen:	Diagnosis length in bytes (max. 250, 0=no diagnosis)

m044_get_slave_diagnosis — Read a slave’s diagnostic data

Pascal	FUNCTION m044_get_slave_diagnosis (micro_slot: byte; slave: byte; size: byte; VAR data) : word;
C	ushort EXPORT m044_get_slave_diagnosis (byte micro_slot, byte slave, byte size, void *data);
Function	This function (27) reads a diagnostic data block (for Octet-1..Octet-6 see standard; other octets (bytes) are user-specific) from a slave’s diagnostic channel.

Parameter	*slave*:	Slave address (3..123)
	size:	Number of bytes for reading
	data:	Address of the destination data buffer
Diagnose	*0*:	Command correctly executed
	100:	Error: bus differently parameterized
	101:	Error: consistency conflict

m044_get_data_transfer_list Read data transfer list

Pascal	FUNCTION m044_get_data_transfer_list (micro_slot: byte; VAR data: m044_slv_bitmap_type) : word;	
C	ushort EXPORT m044_get_data_transfer_list (byte micro_slot, m044_slv_bitmap_type *data);	
Function	This function (28) reads the data transfer list. Each bit in the data read represents one slave (example: Slave 17 = Bit-1 of field slave_16_31). A set bit means that the slave involved is in the DATA (= slave is activated) status.	
Parameter	*data*:	Address of a structure variable of the type m044_slv_bitmap_type (see table below)

The variable type m044_slv_bitmap_type is a data structure consisting of the following fields:

Field	Data type	Significance
.slave_0_15	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_16_31	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_32_47	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_48_63	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_64_79	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_80_95	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_96_111	ushort	Bit field (Bit-0 .. Bit-15 valid)
.slave_112_123	ushort	Bit field (Bit-0 .. Bit-11 valid)

m044_check_slave_active — Check whether slave is active

Pascal	FUNCTION m044_check_slave_active (micro_slot; slave; VAR active: byte) : word;	
C	ushort EXPORT m044_check_slave_active (byte micro_slot, byte slave, byte *active);	
Function	This function (29) checks whether the slave specified is in the DATA (=activated) status.	
Parameter	*slave*:	Slave address (3..123)
	active:	0: Slave is not activated 1: Slave is activated

m044_get_master_status_struct — Read master's status structure

Pascal	FUNCTION m044_get_master_status_struct (micro_slot: byte; VAR data: m044_mst_statusstr_type) : word;	
C	ushort EXPORT m044_get_master_status_struct (byte micro_slot, m044_mst_statusstr_type *data);	
Function	This function (30) reads the master status structure.	
Parameter	*data*:	Address of a master status variable of the type m044_mst_ statusstr_type (see table below)

The variable type m044_mst_statusstr_type is a data structure consisting of the following fields:

Field	**Data type**	**Significance**
.master_status	byte	Master status
.id_high	byte	Hardware Ident Number (High)
.id_low	byte	Hardware Ident Number (Low)
.master_hw_version	byte	Hardware version of the master
.master_fw_version	byte	Firmware version of the master
.user_hw_version	byte	Hardware version of the user
.user_fw_version	byte	Firmware version of the user

m044_get_system_error_struct Read system error structure

Pascal	FUNCTION m044_get_system_error_struct (micro_slot: byte; VAR data: m044_mst_syserr_type) : word;
C	ushort EXPORT m044_get_system_error_struct (byte micro_slot, m044_mst_syserr_type *data);
Function	This function (31) reads the system error structure.
Parameter	*data*: Address of a system error variable of the type m044_mst_ syserr_type (see table below)

The variable type m044_mst_syserr_type is a data structure consisting of the following fields:

Field	**Data type**	**Significance**
.component	ushort	Module name of firmware
.subcomponent	ushort	Module's subcomponent
.status	ushort	Status value
.error_number	ushort	Error number
.detail	ushort	Detail

m044_poll_system_error Read system error by polling

Pascal	FUNCTION m044_poll_system_error (micro_slot: byte; VAR syserr: word) : word;
C	ushort EXPORT m044_poll_system_error (byte micro_slot, ushort *syserr);
Function	This function (32) is used for polling the master's system error (="Component" field of the system error structure). It is required only if the _M044_ENABLE_IRQ parameter has not been set at re-start.
Parameter	*syserr*: "Component" field of the system error structure

Special functions

m044_fw_timer Activate/de-activate firmware timer

Pascal	FUNCTION m044_fw_timer(micro_slot: byte; timer: word) : word;	
C	ushort EXPORT m044_fw_timer(byte micro_slot, ushort timer);	
Function	This function (33) activates or de-activates the master's firmware timer. This (when the interrupt has been activated) cyclically triggers the interrupt set to the base board.	
Parameter	*timer*:	Cycle duration as a factor of 25.6 µs (x 0..65535) *timer* = 0 deactivates the timer
Diagnosis	*0*:	Command correctly executed
	256:	Error: command semaphore not set

Accessing the gate array of the M-DPM-12 module

m044_get_fpga_versionRead version of gate array

Pascal FUNCTION m044_get_fpga_version (micro_slot: byte) : byte;

C byte EXPORT m044_get_fpga_version (byte micro_slot);

Function This function reads the version/revision of the gate array:

Bit-0..Bit-3 : Revision Number

Bit-4..Bit-7 : Version Number

m044_set_modul_register — Set a module register

Pascal	PROCEDURE m044_set_modul_register (micro_slot: byte; reg_type: byte; data: byte);
C	void EXPORT m044_set_modul_register (byte micro_slot, byte reg_type, byte data);
Function	This procedure sets a module register (LED or Interrupt Select Register).

Parameter	*reg_type*:	_M044_ISR:	set Interrupt Select Register
		_M044_LER:	set LED register
	data:	byte to be set (see P. 6-13 for significance)	

m044_get_modul_register — Read a module register

Pascal	FUNCTION m044_get_modul_register (micro_slot: byte; reg_type: byte) : byte;
C	byte EXPORT m044_get_modul_register (byte micro_slot, byte reg_type);
Function	This procedure reads a module register.

Parameter	*reg_type*:	_M044_ISR:	read Interrupt Select Register
		_M044_IST:	read Interrupt Status Register
		_M044_LER:	read LED Register

m044_set_timeout_counter — Set the timeout counter

Pascal	PROCEDURE m044_set_timeout_counter (micro_slot: byte; data: word);
C	void EXPORT m044_set_timeout_counter (byte micro_slot, ushort data);
Function	This procedure sets the module’s timeout counter. This counter is used for consistency control.
Parameter	*data*: value to be set, as a factor of 10 µs (0..65535)

m044_get_timeout_counter — Read timeout counter

Pascal	FUNCTION m044_get_timeout_counter (micro_slot: byte) : word;
C	ushort EXPORT m044_get_timeout_counter (byte micro_slot;
Function	This procedure reads the module's timeout counter. Timeout = return value (0..65535 x 10 µs)

m044_set_cons — Set a consistency request

Pascal	PROCEDURE m044_set_cons (micro_slot: byte; mode: byte);		
C	void EXPORT m044_set_cons (byte micro_slot, byte mode);		
Function	This procedure sets a consistency request.		
Parameter	*mode*:	Read consistency:	1
		Write consistency:	2

m044_clear_cons — Terminate consistency request and read module status

Pascal	FUNCTION m044_clear_cons (micro_slot: byte) : byte;
C	byte EXPORT m044_clear_cons (byte micro_slot);
Function	This procedure terminates the consistency request, and reads the module status (see m044_get_modul_status).

m044_get_modul_status — Read the gate array status

Pascal	FUNCTION m044_get_modul_status (micro_slot: byte) : byte;
C	byte EXPORT m044_get_modul_status (byte micro_slot);
Function	This procedure reads the gate array status (see overview below).

	0	**0**	**0**	**0**	**0**	**0**	**0**	**Module status**
							0	Timeout status 0=no error, 1=error
						0		Consistency status 0=no error, 1=error
					0			Read consistency request 0=active, 1=not active
				0				Write consistency request 0=active, 1=not active
			0					ASPC 2 consistency request 0=not active, 1=active
		0						Host consistency acknowledge 0=active, 1=not active
	0							Host access to DPRAM 0=active, 1=not active

Commissioning

For commissioning the PROFIBUS-DP Master M-DPM-12, you have to transfer the configuration of the system (consisting of master and slaves) in the form of a binary file onto the M-DPM-12 module. The configuration is stored on the module in flash memory, i.e. it is also available after a reset (e.g. due to power-down), and should be transferred only once. If changes are desired in the system configuration, a new binary file must be created and the old configuration on the module overwritten.

The high-level language library supplied (M044_LIB) provides functions for transferring a binary file onto the module (download). The current configuration can also be loaded from the module (upload).

The "COM ET 200" software from Siemens is used to create the configuration file (binary file).

Tips on installation

The Siemens software COM ET 200 must be installed under Windows 3.x. Insert Floppy Disk 1, and call the INSTALL.EXE program. Please remember that when installing under "Options" the memory card drivers have to be de-activated!

The type file for the M-DPM-12 module is supplied on floppy disk together with the M-DPM-12 module ("TYPDATEI" subdirectory). Please copy all files from the \MASTERS\ and \BITMAPS\ subdirectories into the identically named subdirectories of COM ET 200.

Tips on operation (see also COM ET 200 Online Help)

Create a new project (File\New), and the select **Master M-DPM-12** as **Station Number 1**.

You can then, for example, click Siemens ET200 slave stations, and "append" them to the PROFIBUS. Assign a station number, and configure the slave. In the "Configuration" submenu, you need not make an entry in the 'I-Adr.' and 'O-Adr.' columns, since this type of addressing is not supported.

When all slaves have been entered, the bus parameters (e.g. the baudrate) must be set. You can enter the baudrate in the Parameterizing/Bus Parameters menu option. The maximum settable baudrate is limited by the "slowest" slave.

The project can then be saved. A binary file can be created using File\Export.

Programming with the high-level language library

The binary file must be transferred to the M-DPM-12 module with the library function **m044_download_file**. After a download, the module has to be reset. The M-DPM-12 is then in "STOP" status. The ongoing operating mode can be determined with the **m044_get_master_mode** function.

The **m044_set_master_mode** function sets the master to the "OPERATE" status, with the slaves specified in the configuration routine being cyclically addressed. One bit is set in the data transfer list for each active slave. The **m044_check_slave_active** function can also be used to check whether a slave is active. When diagnostic data from a slave are present, a bit is set in the diagnosis list. The **m044_check_slave_ diagnosis** function can be used to interrogate whether a slave has reported diagnostic data.

Data are exchanged between master and slave with the library functions **m044_get_slave_xxx** and **m044_set_slave_xxx**. Note that the user data can be transferred in bytes, words or blocks.

The **m044_poll_system_error** function can be used to determine whether an error has occurred. This function should be called cyclically.

Programming example

Addressing a slave with 4 byte input and 4 byte output user data:

```
/* Set master to operate status               */
 m044_set_master_mode(microslot, _M044_OPERATE);
 m044_refresh_slave_datastruct(microslot);

/* Check whether slave is active              */
 m044_check_slave_active(microslot, slave, &active);
 if(active)
  {
/* Exchange user data (4 bytes each)          */
   m044_set_slave_data_block(microslot, slave, 4, &dout);
   m044_get_slave_data_block(microslot, slave, 4, &din);
  }
/* Check whether slave has reported diagnosis */
m044_check_slave_diagnosis(microslot, slave, &diaglen);
if(diaglen > 0)
  {
   m044_get_slave_diagnosis(microslot, slave, diaglen, &diag);
  }
```

Programming with I/O access operations

This chapter is intended for those users who want to write their own application programs for the MODULAR-4/486 base board.

Local I/O addresses

All addresses are written in hexadecimal form. Bits not used are reserved, and should be set to 0 when writing. These bits are not valid during reading.

Address	**Access**[1]	**Function**
MBA+00h	RW16	Set/read DPRAM pointer (0000h-3fffh) Bit 0 is used to select low or high-byte for byte access operations to the DPRAM (address line A0).
MBA+03h	RW8	LED Register (LER) During reading, Bits 0 and 1 specify the status of LEDs 1 and 2 (0 = off, 1 = on). During writing, Bits 0 and 1 select LEDs 1 and 2, while Bits 2 and 3 specify the value to be set.

[1] R16: 16 bit read access, W16: 16 bit write access, RW16: 16 bit read or write access, R8: 8 bit read access, W8: 8 bit write access, RW8: 8 bit read or write access, W8x = 8 bit write access, any data

Address	**Access**[1]	**Function**
MBA+01h	RW8	Interrupt Select Register (ISR) / conflict detection Bit 0 to 2 determine the interrupt line to the base board: 0 0 0 no interrupt line selected 0 0 1 IRQ-A 0 1 0 IRQ-B 0 1 1 IRQ-C 1 0 0 IRQ-D 1 0 1 IRQ-E 1 1 0 IRQ-F 1 1 1 NMI Bit 4 Interrupt through XINTH (1 = enable) Bit 5 specifies the response to consistency conflicts: 0 Timeout counter active (no interrupt) 1 Interrupt when A-CONS = 1 & XHKAK = 0 Bit 6 Interrupt per firmware timer (1 = enable)
MBA+05h	R8	Interrupt Status Register (IST) A set bit (=1) signals an interrupt: Bit 0 XINTH interrupt active Bit 1 KSTATUS interrupt active Bit 2 Firmware timeout interrupt active

MBA+05h	W8	Reset PROFIBUS master
		Data = 8fh activated XRESET
		Data = 0fh de-activated XRESET

MBA+07h	R8	Read gate array version Bits 0 to 3 = Revision Number, Bits 4 to 7 = Version Number e.g.: 0010 0101 = 25h = Version 2, Revision 5
MBA+07h	W8x	Execute gate array and PROFIBUS master reset All registers are set to 0. The timeout counter is set to ffffh.
MBA+08h	RW16	Set/read timeout counter TOR (16 bits)

Address	**Access**[1]	**Function**
		You can set values 0 to 65535 (x 10µs).
MBA+10h	RW16	Read/write DPRAM-word
MBA+12h	RW16	Read/write DPRAM-byte[1]
MBA+14h	RW16	Read/write DPRAM-word, after which the address pointer will be incremented by 2
MBA+16h	RW16	DPRAM-Byte read/write, after which the address pointer will be incremented by 1

MBA+18h	W8	Set/erase consistency request
		Data = 00h: Erase consistency request and module status
		Data = 01h: Read consistency (XRHCONS = 0)
		Data = 02h: Write consistency (XWHCONS = 0)

MBA+18h	R8	Terminate consistency request and read module status Bit 0 = TIMEOUT STATUS (0=OK, 1=ERROR) Bit 1 = CONSISTENCY STATUS (0=OK, 1=ERROR) Bit 2 = R-CONS (read consistency request, low-active) Bit 3 = W-CONS (write consistency request, low-active) Bit 4 = A-CONS (ASPC2 consistency request) Bit 5 = XHKAK (host consistency acknowledge, (low-active) Bit 6 = XCSDPR2 (host access to DPRAM, (low-active)
MBA+19h	R8	Read module status see MBA+0x18h

[1] R16: 16 bit read access, W16: 16 bit write access, RW16: 16 bit read or write access, R8: 8 bit read access, W8: 8 bit write access, RW8: 8 bit read or write access, W8x = 8 bit write access, any data

[1] In the case of byte accessing to the DPRAM, Bit 0 of the address pointer selects whether the low or the high byte is used (Bit 0 = 0: Lowbyte, Bit 0 = 1: Highbyte). The base board must, however, execute word access operations.

Index for M-DPM-12