# Table of Contents

1 System configuration............................................................................................................. 3
2 Profibus basic concepts ........................................................................................................ 4  
  2.1 DP-Cycle....................................................................................................................... 4  
  2.2 Slave user data and configuration.................................................................................. 5  
  2.3 Baudrate an length of bus cables.................................................................................. 5  
  2.4 Profibus data transfer times.......................................................................................... 5  
3 Set-up of VME-PROF-M........................................................................................................ 6  
  3.1 Installation of the VME-Prof-M.................................................................................... 6  
  3.2 Connecting the PC.......................................................................................................... 6  
  3.3 GSD files........................................................................................................................ 7  
  3.4 Setting up of Slave Parameters..................................................................................... 7  
  3.5 Set up of the Master Parameter..................................................................................... 9  
  3.6 Save the set-up............................................................................................................... 10  
  3.7 Test of the Configuration............................................................................................. 10  
  3.8 Saving the Configuration in the VME-PROF-M............................................................. 11  
  3.9 Replacement in service case.......................................................................................... 12  
4 VMEbus integration................................................................................................................ 12  
  4.1 Reset and Enable............................................................................................................ 12  
  4.2 Operating supervision..................................................................................................... 13  
  4.3 Data exchange without consistence............................................................................... 13  
  4.4 Data access with consistence solution ......................................................................... 14  
  4.5 Interrupt Operating........................................................................................................ 15  
  4.6 Profibus Diagnosis......................................................................................................... 16  
  4.7 Bus Stop and Watchdog............................................................................................... 16  
  4.8 Bus Management Commands....................................................................................... 17  
  4.9 Configuration via VMEbus............................................................................................. 17  
5 Tables and additional information........................................................................................ 18  
  5.1 Technical Specifications............................................................................................... 18  
    5.2 Known errors of the VME-PROF-M Firmware............................................................. 18  
    5.3 Status-LED’s............................................................................................................... 19  
    5.4 PROFIBUS-Connector X2........................................................................................ 19  
    5.5 PC-Connector X 1...................................................................................................... 20  
    5.6 VMEbus-Interface....................................................................................................... 21  
    5.7 Dual-Port-RAM Address Map.................................................................................... 21  
6 Example programs................................................................................................................. 24  
  6.1 ANSI-C example program.............................................................................................. 24  
    6.1.1 DPSTART.C........................................................................................................... 24  
    6.1.2 PROFM.C............................................................................................................... 29  
    1.1.1 PROFM.H............................................................................................................... 37  
  6.2 Example for Berghof VME-S5 CPU............................................................................. 38  
  6.3 Example for an Adept system....................................................................................... 39
1 System configuration

The following pictures show a typical system configuration with the VME-PROF-M in a VMEbus system to connect Profibus I/O modules. A PC is connected via RS 232 com port for configuration purpose.

Using the VME-PROF-M enables you to use cost-saving Profibus I/O modules to replace expensive VMEbus I/O boards. A further advance is the easier installation in the rack and in the factory floor, also service and storage of replacement parts is easier.

Because of the time synchronous DP-V2 data transfer the VME-PROF-M is also an ideal solution for connecting drives.
2 Profibus basic concepts

The Profibus is an European standard field bus system specified in EN 50170. The VME-PROF-M works with the Profibus DP protocol. Profibus-DP is a communication protocol for a controller connected with a group of decentral I/O devices.

2.1 DP-Cycle

The PLC controller (the Profibus Master) exchanges data with the I/O devices in a cycle way. Data are send to all slaves one after the other. The master sends output data to the slave and the slave responds with its input data. After this it is going on to the next slave. When all slaves have been used the cycle begins from the start. This whole cycle is called DP-cycle.

The VME-PROF-M starts every DP-cycle with a „Global Control Telegram“. This can be used for synchronization the slaves. At the end of the slave data exchange a VMEbus interrupt could be generated. From this time on the VMEbus CPU can use the Dualport-RAM of the VME-PROF-M for reading and writing new data. The VME-PROF-M uses this time slot for management services on the Profibus (GAP Update and Token passing to other masters).

The DP cycle can run free (DP-V0) or can run in a fixed timing (DP-V2). We recommend to use always a fixed cycle time. The VME-PROF-M controls the cycle in this case with a hardware timer at microseconds precision.
2.2 Slave user data and configuration

The Masters sends an output data telegram with a maximum length of 244 Bytes. The slave responds with an input data telegram also with a maximum length of 244 Bytes. The actual data length depends on the slave requirements. Profibus-DP supports modular slave stations. This includes a buscoupler and more different I/O modules. The actual slave configuration is described in configuration bytes. The master checks the agreement of the actual configuration and the configuration bytes at start-up. Only when the agreement is O.K. The slave goes in working mode.

A simple slave for example could have the following configuration:

1 analog input module with 4 channels each 16 Bits = 4 input words = 8 Byte
1 digital input module with 8 channels each 1 Bit = 1 Byte input
1 digital output module with 8 channels each 1 Bit = 1 Byte output

Together we have 1 Byte output and 9 Byte input what is totally 10 Byte user data.

2.3 Baudrate and length of bus cables

The Profibus works with a fixed baud rate controlled by the master. The slaves automatically switch to this baudrate. The Baudrate can be selected for matching to the slowest slave and the length of the bus cable.

The following table shows the possible cable length on different baud rates for a standard confirm bus cable Type A:

<table>
<thead>
<tr>
<th>Baudrate kbit/sec</th>
<th>19.2</th>
<th>93.75</th>
<th>187.5</th>
<th>500</th>
<th>1500</th>
<th>3000</th>
<th>6000</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Length m</td>
<td>1200</td>
<td>1200</td>
<td>1000</td>
<td>400</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Stub lines m</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit time ysec</td>
<td>52</td>
<td>10.6</td>
<td>5.3</td>
<td>2</td>
<td>0.66</td>
<td>0.33</td>
<td>0.167</td>
<td>0.083</td>
</tr>
<tr>
<td>Byte time ysec</td>
<td>572</td>
<td>116.6</td>
<td>58.3</td>
<td>22</td>
<td>7.3</td>
<td>3.6</td>
<td>1.8</td>
<td>0.9</td>
</tr>
</tbody>
</table>

2.4 Profibus data transfer times

With the sum of all user data from the slave devices and the baud rate you can calculate the Profibus data transfer time.

The Global Control Telegram contains 13 Bytes data and 3 Idle Bytes.

For the protocol frame and the timing each slave requires 25 Bytes.
Profi_time = Byte_time * ( 15 + 25 * Number_of_slaves + Sum_Number_of_User_bytes )

Example: 1,5 MB/sec → Byte_time = 7,3 ysec

5 Slaves each 10 Byte User_data

Profi_time = 7,3 * (15 + 25 * 5 + 5 * 10 )
= 7,3 * 190 = 1387 ysec

For the setup of the DP cycle you should add 30% for safety.
So 2 ms is a useful value. Then you have 0,6 ms for the CPU time.
If the CPU time is not enough or other masters (diagnostic equipment) use the bus, the time has to be increased for reaching a robust operation.

3 Set-up of VME-PROF-M

For setting up a VME-PROF-M you need
a PC with Windows XP and the DPKonfig software
1 free com Port (RS-232)
1 Zero-Modem RS 232 cable with Dsub connectors (Pin 2 and 3 crossed over)
the GSD files of all installed Slave devices
and of course the VMEbus system.

For start-up the VME-PROF-M you don't need any software on the VMEbus CPU at this time.

3.1 Installation of the VME-Prof-M

The VME-PROF-M needs a 64 kByte address space in the VME standard address room. The start address is set by HEX switches S1 and S2.
S2 is the MSD (A23..A20) and S 1 the LSD (A19..A16).
Example: address 0x85000 set S2 on 8 and S1 on 5
Please set the run jumper, so VME-PROF-M starts immediately after the Reset. Please switch the VMEbus system off before you start mounting the VME-PROF-M. Please use both screws on the front panel to make sure the grounding.

3.2 Connecting the PC

Please install the DPKonfig software on your PC. You have to start Setup.exe in the directory Disk1 of the installation CD by double clicking. At the end of the installation procedure you have to copy the DPKonfig.exe from the CD in your directory C:\DMS\CMDS.
Now connect the COM1 or the COM2 port of your PC with the cable to the X1 connector of the VME-PROF-M. Please do not connect the Profibus at this time.

Switch on the VMEbus system.

Start the DPKonfig on your PC and select the point “VME-Profi” on COM1 or COM2 from the menu. In the message line you should see now; “VMEprofi on COM firmware version 8.0”

### 3.3 GSD files

For each slave device the manufacturer supplies a GSD file. For use with DPKonfig you should copy this file in the directory C:\DMS\GSD. The name conventions are also required. A GSD name starts with 4 letters – they came from the manufacturer name (SIEM for Siemens). Then follow 4 hex digits for the ident number of the slave. The file suffix has always to be GSD. Please rename the GSD file if its required.

For example: SIEM000E.GSD

### 3.4 Setting up of Slave Parameters

Select in the main screen of DPKonfig a slave address – for example number 10.
Now click the “Slave” Button to open the Slave editor screen:

Please press the “New” button and then the “GSD” button. Now you can select the GSD file. For example WAGOB750.GSD.

The main data from the GSD file will be displayed now. On the top of the screen between station address and ID-number you can add a name for your station, for instance “MessStation”.

By using the select box on the right side of the screen you get the installed modules, for example a 4 channel PT 100 input, a 8 channel digital input and a 8 channel digital output. Be sure to add the modules in the correct order. Your selection and the hardware has to be the same. It is useful to put modules with word data (16 Bit) first, so they have even addresses assigned in the Dual-Port-RAM.

For each module one configuration byte is generated.

If you have entered all modules please press the Auto address button to generate the DP run address offsets. As a result you see on the left side in the example 9 Byte input data at DPR offset 0 and 1 Byte output data at DPR offset 0x800. Those are your 10 user bytes.

For each slave you can set a watchdog time. In case of a bus or master failure the slave switches all outputs to 0 – if this time is elapsed.

Press o.k. now and continue this process for all slaves.
3.5 **Set up of the Master Parameter**

Please press the “Master” button now to open the Master Window.

First set up the Baud rate. See capt. 1.3 (for instance 1,5 Mbaud).

With the DP parameter “MinSlaveInterval” you set up the DP cycle time (for example 10 ms).

It is recommended to take a big value on start and optimize it later. If you would take a too small value, the Profibus runs free without a fixed timing.

The Slot time is the time window for a slave which sends no answers. This time should be set with the same time as a correct answering slave is using. The time could be entered in bit times and could be displayed in milliseconds by clicking the check box “ms”.

With the parameter data control time the monitoring time for the slave communication can be set. The value should be greater than 2 * DP cycle and less than 0,5 * Watchdog time of the slave.

If a slave does not receive data within this time window it is reported as a failure. The life bit of this slave in the Dual-Port-RAM is cleared to report the failure to the main CPU in the VMEbus system.

When the option “Autoclear” is selected the VME-PROF-M also sets the outputs of all other slaves to Zero and goes in the operating mode “clear”.

For all other parameters you can use the default values – changing is only required in multi-master-systems or if you are using repeaters.

Close the master window with the OK button.
3.6 **Save the set-up**

Save your configuration by using the File menu and the command “Store” on the hard disk in a DP1 file. With the command “Print” you can print out a documentation of your configuration.

3.7 **Test of the Configuration**

By Pressing the button “Run” you can transfer your configuration into the VME-PROF-M. After the transfer the VME-PROF-M starts operating. The red LED over the Profibus connector switches off and the yellow LED starts lighting.

**Attention: Danger!**

Before you can connect the Profibus cable be sure that your plant cannot be damaged through false output data.

Is the Profibus connected now, the green LED shows the “receive data”. The slaves now will be initialized and when the configuration is correct and equal to the real hardware the slave enters the data exchange phase. The DPKonfig now shows you a bus connection line.
Shown on the picture: Slave 10 is ready

Slave 12 has a problem. By Clicking in the Main Window it will be selected and its diagnostic could be read out. The message “0 diagnostic bytes” tells you that the slave is not connected to the bus or it has no operating power or its address is switched to a wrong number.

Our slave 12 shows 17 diagnostic bytes. With a click on the right mouse button you can open a diagnostic window to interpret the Profibus diagnostics.

You can see that the reason is “Cfg. Fault”.

![Station 12 Diagnose](image)

### 3.8 Saving the Configuration in the VME-PROF-M

Please do not forget to save the tested version of your configuration on your hard disk (see capt. 3.6).

Now use the menu EEPROM to put the configuration in the EEPROM of the VME-PROF-M. The EEPROM stores the configuration also if power goes off. From this time on the VME-PROF-M starts after reset with this configuration.

Click on the EEPROM menu at first “Stop” and then “Burn”. The message “programming ok” is shown on success. You are also able to read back a configuration from the EEPROM.
3.9 Replacement in service case

If there is the need to change a VME-PROF-M in a plant, you have to re-install the configuration in the new VME-PROF-M. For this purpose use your saved DP1 file. If possible you also can read the configuration from the old VME-PROF-M with the EEPROM menu. The GSD files of the slaves are not needed.

Menu File ->Load xxx.DP1
Menu Hardware ->VMEprofi->Com1/2
Menu EEPROM → Burn

4 VMEbus integration

When the VME-PROF-M is configured ready with the DPKonfig and the configuration data are stored in the EEPROM, the VME-PROF-M can start automatically the Profibus system. The integration into the VME system is now very easy and you do not need much software. An important question is the data consistency by getting data from the Dual-Port-RAM. There are three possibilities discussed in capt. 4.3, 4.4 and 4.5.

4.1 Reset and Enable

In a plant the Profibus should only be started after enabling by the VMEbus CPU. The Run Jumper on the VME-PROF-M has to be removed.

After the VMEbus Reset the red Sys-Fail-LED is on and the Profibus stops. Enabling is done by writing 0x08 into the control register (DPR offset 0x000E). The Profibus now starts – the red LED is off.

```c
printf("Start program for VME-PROF-M \n") ;
printf("---------------------------- \n
") ;

/* Set Pointer to VMEbus base */
p = (byte*) 0x870000 ;

DP = (VS_Typ*) p ; /* DPR of the VME-PROFI */
Profi = (ProfiDPR_Typ*) (p + 0x1000) ; /* Pointer to PROFIBUS data */
IRQflag = (wort*) (p + 0xFFFF) ; /* Pointer to IRQ Flag Register */

/* Request the accessment rights from the Operating system */
if ( (err = _os9_id(&pid,&sink,&sink,&sink)) )
    printf("Error _os9_id() %X\n",err);
else
    printf("Request accessment rights success!")  ;
if ( (err = _os_permit ((void*)p,0x1000,0x777,pid)))
```
printf("Error _os_permi() %X \n",err);

/* print Board-Ident */
printf(" Manufacrurer-Ident ...: %c%c%c\n", DP->ID1,DP->ID2,DP->ID3 );
printf(" Board-Ident .........: %3d\n", DP->BoardID );
printf(" Board-Version ........: %3X\n", DP->BoardED );

/* Init of the VME-PROFI */
printf("Init VME-PROFI : \n") ;
DP->Control = 0x00 ; /* Reset VME-PROF-M */
WaitMS (100) ;
DP->Error = 1 ; /* Pre-value, will be cleared from the VME-PROF-M */
DP->Comand = 1; /* Pre-value, will be cleared from the VME-PROF-M */
DP->Control = 0x08 ; /* 08 = enable VME-Prof-M */

4.2 Operating supervision
The VME-PROF-M has to Live counters:

Live (DPR Offset 0x040E is incremented when the locale CPU in the VME-PROF-M is running.

Proficycle (DPR Offset 0x2FEC) counts the DP-Cycles.

The Bit field DPM1Live contains one bit for each Slaveaddress. This bit is set when the data exchange is o.k.

/* DP_StationRun shows the status of the slave station
   The station address must be passed.
   Return value: 1 if the slave is present at the bus and data are
   transfered (slave is running)
   0 not running */

short DP_StationRun ( byte a )
{
    byte n , b ;
    n = a >> 3 ;
    b = 1 << (a & 7) ;
    if (Profi->DPM1Live[n] & b) return 1 ;
        else return 0 ;
}

4.3 Data exchange without consistence
Data access to bytes or words (16 bit) with even addresses will be always consistent. This will be done by the Dual-Port-RAM arbitration. For Profibus systems with simple I/O modules (digital, analog I/O) no extra data consistence solutions are required if the configuration assigns word data to even
addresses. In most cases this could be done by installing the analog modules at first (close to the buscoupler).

### 4.4 Data access with consistence solution

By access on data of more than 16 Bit are additionally procedures required to grant data consistence. The VME-PROF-M has 2 semaphores.

Profi->OutDataState has to be incremented from the user program before it writes data.

In Profi->OutDataLastWrite the Slaveaddress has to be written. After writing the user data Profi->OutDataState has to be incremented once more. This procedure should run as fast as possible and may not be interrupted from other interrupt procedures or task changes.

In fact the VME-PROF-M is blocked for this time.

```c
/* Profi_WriteDaten copies the output data of a slave station from the memory (pointer *OutDaten) into the DPR.  
The station address must be passed to ensure data consistency.  
Offset is the station's Outputoffset within the DPR  
Anzahl is the output data length / bytes */

void Profi_WriteDaten (byte Station, wort Offset, wort Anzahl, byte *OutDaten ) {
    wort n ;  
    Profi->OutDataState ++ ;  
    Profi->OutDataLastWrite = Station ;  
    for (n=0;n<Anzahl;n++) Profi->DPMlio[Offset+n] = OutDaten[n] ;  
    Profi->OutDataState ++ ;
}
```

By reading data the user program first saves Profi->InDataState. Then the user data could be read. After reading, the program has to be checked, whether there was a change in Profi->InDataState.

If this is the case and it is the actual slave you have to repeat the reading procedure.

```c
/* Profi_ReadDaten copies the input data of a slave station from the DPR into the memory (pointer *InDaten)  
The station address must be passed to ensure data consistency.  
Offset is the station's Inputoffset within the DPR  
Anzahl is the input data length / bytes */

void Profi_ReadDaten (byte Station, wort Offset, wort Anzahl, byte *InDaten ) {
    wort n ;  
    byte old, lastStation, diff ;
}
Loop:

old = Profi->InDataState;
for (n=0; n<Anzahl; n++) InDaten[n] = Profi->DPM1io[Offset+n];
lastStation = Profi->InDataLastWrite;
diff = Profi->InDataState - old;
if (old & 1)
  { /* ungerade */
    if (diff > 1) goto Loop;
    if (Station == lastStation) goto Loop;
  }
else
  { /* gerade */
    if (diff > 2) goto Loop;
    if ((diff == 1) && (Station == lastStation)) goto Loop;
  }

4.5 Interrupt Operating (starting from Software Version 8.0)

Operating with interrupts is the best solution. The VME-PROF-M has to be configured for time synchronous operation (MinSlaveInterval=DP-cycle). The VME-PROF-M generates a VMEbus interrupt after completing the data exchange with all slaves.

In the interrupt service routine there is a defined time slot for the VMEbus CPU to access the data in the Dual-Port-RAM. No data consistence problems exist.

Initializing in the main program – Example Vector 95 Level 2:

n = irqinst (ProfiIRQ, 95, 2); /* Install IRQ Service for OS-9 */
DP->Vector = 95;            /* Set VME-PROF-M Vertor Register */
DP->Control = 0x08 + 2;     /* Enable IRQ Level 2 */

The data processing is done in an interrupt service routine which is called from the VME-PROF-M in a fixed time interval (DP-cycle) periodically. On every time when the interrupt service routine is called, actual slave input data are in the Dual-Port-RAM. The interrupt routine can read the data and start a high prior task for processing or can do all needed processing.

The the end of the CPU time slot is the latest time for new output data to be written in the Dual-Port-RAM. The VMEbus CPU now can process other jobs while the VME-PROF-M transfers the I/O-data to the slaves.

/* ----------- Interupt Service ---------------------- */
int ProfiIRQ (void)
{
  if (*IRQflag)
  {
    *IRQflag = 0;            /* Cler IRQ flag */
  }
Wert = Profi->DPM1io[0] ; /* Input Byte read */
Wert++ ; /* if Inputs and Outputs on the Slave are connected the Count demo runs */
Profi->DPM1io[0x800] = Wert ; /* Output Byte write */
return 1 ;
}
return 0 ;

The actual consumed time for the Profibus data transfer (T-profi) can be read from the Dual-Port-RAM variable Profi>ProfiCTime. The CPU time window you can calculate: Tcpu=Dpzyklus-Tprofi.

Dpzyklus is set up from the master parameter “MinSlaveInterval”.

4.6 Profibus Diagnosis

If required diagnostic messages from slave stations can be treated.

In the bit field DPM1Diag is one Bit for each slave, which has a diagnostic message.

/* DP_StationDia shows the if diagnostic data are available for this slave
   The station address must be passed.
   Return value: 1 diagnostic data available
   0 no diagnostic data */
short DP_StationDia ( byte a )
{
    byte n , b ;
    n = a >> 3 ;
    b = 1 << (a & 7) ;
    if (Profi->DPM1Diag[n] & b) return 1 ;
    else return 0 ;
}

If a diagnostic message exists, it can be read with the “GetDiagnose” service.

Please see the software example:

4.7 Bus Stop and Watchdog

A VMEbus Reset or Writing the Control Register (DPR Offset 0x000E) to Zero sets the VME-PROF-M on Reset (Halt). No more Profibus telegrams are send or received.
For the slaves this looks like a Profibus failure or a broken cable. Each slave sets its outputs zu 0 (after the watchdog time is elapsed).

The watchdog time was configured in the slave menu of the DPKonfig.

**4.8 Bus Management Commands**

The VME-PROF-M has a command interface in the Dual-Port-RAM for Profibus management services. The most users do not need this because VME-PROF-M does automatically the Profibus initialization via EEPROM parameters.

The following services are implementend:

- Set operating mode  ->  Offline  Stop  Clear  Operate
- LoadMasterParameter
- LoadSlaveParameter
- GetDiagnose
- GlobalControl  ->  Freeze, UnFreeze, Sync, UnSync

For application examples please see the demo program in capt. 6

**4.9 Configuration via VMEbus**

In some plants it is required to hold all configuration data on a single data base. For those applications it is possible to store the configuration file that was generated by the DPKonfig in the file system of the main CPU. The EEPROM of the VME-PROF-M has to be erased for being empty. A program running on the main CPU now can control the start-up phase of the Profibus with a xx.DP1 data over the DP-RAM of the VME-PROF-M.

A sample application you can find in the procedure DPrun of the ProfM.c example capt.6.
5 Tables and additional information

5.1 Technical Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMEbus:</td>
<td>A24, D16 Slave</td>
</tr>
<tr>
<td>Memory:</td>
<td>64KB Dualport-RAM</td>
</tr>
<tr>
<td></td>
<td>128 KB RAM</td>
</tr>
<tr>
<td></td>
<td>512 KB Bootblock Flash EPROM</td>
</tr>
<tr>
<td>Interface:</td>
<td>1 * RS485 PROFIBUS, galvanic separated, 12 MBit/sec with PROFIBUS-FPGA</td>
</tr>
<tr>
<td></td>
<td>1 * serial RS232</td>
</tr>
<tr>
<td>Processor:</td>
<td>32-Bit</td>
</tr>
<tr>
<td>Protocol:</td>
<td>PROFIBUS DP-Master 1 ( EN 50170)</td>
</tr>
<tr>
<td>Dimension:</td>
<td>VME-PROF-M 6HE-4TE (3HE board with 6HE frontplate)</td>
</tr>
<tr>
<td></td>
<td>VME-PROF-M3 3HE-4TE (single euro format, 160 x 100 mm)</td>
</tr>
<tr>
<td>Power:</td>
<td>+5V: typ. 700 mA</td>
</tr>
<tr>
<td></td>
<td>+5V Standby not needed</td>
</tr>
<tr>
<td></td>
<td>+12V and -12V not needed</td>
</tr>
<tr>
<td>Temperature:</td>
<td></td>
</tr>
<tr>
<td>Operating:</td>
<td>0 ... +50 °C</td>
</tr>
<tr>
<td>Storage:</td>
<td>-25 ... +70 °C</td>
</tr>
</tbody>
</table>

5.2 Known errors of the VME-PROF-M Firmware

Limitation for all versions:
Configuration with a Profibus Class 2 master is not possible

Errors in version 7.2
Only Single-Master operation possible.
DP-Variable ProfiCtime is not valid

Errors in version 8.0
No errors known
5.3 Status-LED’s

6 LED’s on the front panel show the operating mode:

1. LED-Row (VMEbus Status)
   - **green** (A activ) VMEbus access to the VME-PROF-M
   - **yellow** (I Interrupt) The VME-PROF-M generates a VMEbus-IRQ
   - **red** (F Fail) The VME-PROF-M is not yet enabled

2. LED-Row (PROFIBUS Status)
   - **green** (R RxData) The VME-PROF-M receives Data from the PROFIBUS
   - **yellow** (T TxData) The VME-PROF-M sends on the PROFIBUS
   - **red** (F Fail) Off: The PROFIBUS-Controller runs
     - not enabled
     - no Busparameters
   - On: The VME-PROF-M has stopped (halt)

5.4 PROFIBUS-Connector X2

The connector X 2 is a norm conform Profibus connector. All signals are potential isolated from the VME system.

<table>
<thead>
<tr>
<th>Connector X2</th>
<th>I/O - Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 1</td>
<td>n.c.</td>
</tr>
<tr>
<td>Pin 2</td>
<td>n.c.</td>
</tr>
<tr>
<td>Pin 3</td>
<td>Daten B (RxDTxD-P)</td>
</tr>
<tr>
<td>Pin 4</td>
<td>CNTR (Repeater control signal TTL)</td>
</tr>
<tr>
<td>Pin 5</td>
<td>GND</td>
</tr>
<tr>
<td>Pin 6</td>
<td>Termination-Power</td>
</tr>
<tr>
<td>Pin 7</td>
<td>n.c.</td>
</tr>
<tr>
<td>Pin 8</td>
<td>Daten A (RxDTxD-N)</td>
</tr>
<tr>
<td>Pin 9</td>
<td>n.c.</td>
</tr>
<tr>
<td>Connector Case</td>
<td>Shield</td>
</tr>
</tbody>
</table>
5.5 **PC-Connector X1**

On the connector X1 you find a RS232 Com-Port for the connection of the PC. The Pin-out is PC-conform. Only the signals RxD, TxD and GND are used. The port uses the following parameters:
9600 Baud, 1Stop Bit, No-Parity, Terminal Emulation ANSIVT100.
You need a Null-Modem-cable with 2 female 9-pin connectors (Pin 5 connected, Pin 2 and 3 crossed over).

**Note:**
There is no potential separation on this port.
The port is only used for test and configuration purpose.

<table>
<thead>
<tr>
<th>Pin</th>
<th>RS232</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>RXD Receive data</td>
</tr>
<tr>
<td>3</td>
<td>TXD Transmit data</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Do no connect</td>
</tr>
<tr>
<td>8</td>
<td>Do not connect</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
</tr>
</tbody>
</table>

*Figure 1: Cable connection VME-PROFI X1 - PC (VT100 Terminal)*
WARNING

The configuration of input/output bytes to the dualport RAM addresses can be changed from the DPKonfig. That could cause wrong settings of output bytes or reading wrong input bytes.

Additionally output bytes can be set directly.

Never use DPKONFIG to control VME-PROF-M, if connected machines are running!!

5.6 VMEbus-Interface

The table below summarizes a number of VMEbus functions:

<table>
<thead>
<tr>
<th>VMEbus-Function</th>
<th>Included</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTB Master Address</td>
<td>-</td>
<td>no VMEbus-Master functions</td>
</tr>
<tr>
<td>Master Data Bus</td>
<td>-</td>
<td>&quot;</td>
</tr>
<tr>
<td>Slave Address Bus</td>
<td>A24</td>
<td>24 Bit Address Bus width</td>
</tr>
<tr>
<td>Slave Data Bus</td>
<td>D08, D16</td>
<td>8 and 16 Bit Data bus width</td>
</tr>
<tr>
<td>SSEQ</td>
<td>Np</td>
<td>Slave: Sequential Access</td>
</tr>
<tr>
<td>Interrupt-Handler</td>
<td>-</td>
<td>No Interrupt-Handler function</td>
</tr>
<tr>
<td>Interrupter</td>
<td>IH1, IH2, IH3, IH4, IH5, IH6</td>
<td>Programmable IRQ-Level</td>
</tr>
</tbody>
</table>

5.7 Dual-Port-RAM Address Map

<table>
<thead>
<tr>
<th>Address (Hex)</th>
<th>Type</th>
<th>Name</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMEbus-Standard base address +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0000</td>
<td>Byte</td>
<td>Dummy0</td>
<td>Dummy</td>
</tr>
<tr>
<td>$0001</td>
<td>Byte</td>
<td>ID1</td>
<td>&quot;D&quot;</td>
</tr>
<tr>
<td>$0002</td>
<td>Byte</td>
<td>IRQSET</td>
<td>reserved</td>
</tr>
<tr>
<td>$0003</td>
<td>Byte</td>
<td>ID2</td>
<td>&quot;M&quot;</td>
</tr>
<tr>
<td>$0004</td>
<td>Byte</td>
<td>IRQEN</td>
<td>reserved</td>
</tr>
<tr>
<td>$0005</td>
<td>Byte</td>
<td>ID3</td>
<td>&quot;S&quot;</td>
</tr>
<tr>
<td>$0006</td>
<td>Byte</td>
<td>Dummy3</td>
<td>Dummy</td>
</tr>
<tr>
<td>$0007</td>
<td>Byte</td>
<td>BoardID</td>
<td>S0B</td>
</tr>
<tr>
<td>$0008</td>
<td>Byte</td>
<td>Dummy4</td>
<td>Dummy</td>
</tr>
<tr>
<td>Address</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>$0009</td>
<td>Byte</td>
<td>BoardED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Board Vers. = &lt;D7..D4&gt;,&lt;D3..D0&gt;</td>
<td></td>
</tr>
<tr>
<td>$000A</td>
<td>Byte</td>
<td>Dummy5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dummy</td>
<td></td>
</tr>
<tr>
<td>$000B</td>
<td>Byte</td>
<td>Mail</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$000C</td>
<td>Byte</td>
<td>Dummy6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dummy</td>
<td></td>
</tr>
<tr>
<td>$000D</td>
<td>Byte</td>
<td>Vector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$000E</td>
<td>Byte</td>
<td>Dummy7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dummy</td>
<td></td>
</tr>
<tr>
<td>$000F</td>
<td>Byte</td>
<td>Control</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Card-Control-Register</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>By writing $08 into this register, the VME-PROFI will be enabled</td>
<td></td>
</tr>
<tr>
<td>$0010</td>
<td>Lword[252]</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$0400</td>
<td>Byte</td>
<td>Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0: load EEPROM config data)</td>
<td></td>
</tr>
<tr>
<td>$0401</td>
<td>Byte</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$0402</td>
<td>Word[6]</td>
<td>Prm</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$040E</td>
<td>Word</td>
<td>Live</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>“live“ will be increased while the VME-PROFI is running</td>
<td></td>
</tr>
<tr>
<td>$0410</td>
<td>Byte[256]</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$0510</td>
<td>Byte[256]</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$0610 - $070F</td>
<td>Word[128]</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$0710</td>
<td>char[7]</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>„DPM1“ = DP-Master1 software</td>
<td></td>
</tr>
<tr>
<td>$0717</td>
<td>char</td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SW-version = &lt;D7..D4&gt;,&lt;D3..D0&gt;</td>
<td></td>
</tr>
<tr>
<td>$0718 - $0FFF</td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$1000-$1FFF</td>
<td></td>
<td>DPM1io</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 KB I/O-range</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: inputs starting at $1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>outputs starting at $1800</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The I/O-range can be arbitrary</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>configured by the user</td>
<td></td>
</tr>
<tr>
<td>$2000-$200F</td>
<td>Byte[16]</td>
<td>DPM1Live</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit-field variable, 1 Bit per Slave address (0 -127). The Bits are set, in the case, that the corresponding slave answered positive. Slave-Adr. 0 -&gt; $2000 LSB (D0) Slave-Adr. 7 -&gt; $2000 MSB (D7) Slave-Adr. 127 -&gt; $200F MSB (D7)</td>
<td></td>
</tr>
<tr>
<td>$2010-$201F</td>
<td>Byte[16]</td>
<td>DPM1Diag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit-field variable, 1 Bit per Slave address. The bits are set, if diagnostic data for the corresponding slave are available Slave-Adr. 0 -&gt; $2010 LSB (D0)</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>$2020</td>
<td>Byte</td>
<td>Remote AccessFlag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved (Access flag for class 2 Master)</td>
<td></td>
</tr>
<tr>
<td>$2021</td>
<td>Byte</td>
<td>InDataState</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flag for input consistency, increased before and after writing of the input data (read only)</td>
<td></td>
</tr>
<tr>
<td>$2022</td>
<td>Byte</td>
<td>InDataLastWrite</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slave address of the last written input data (read only)</td>
<td></td>
</tr>
<tr>
<td>$2023</td>
<td>Byte</td>
<td>OutDataState</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flag for output consistency, increased before and after writing of the output data (set by user)</td>
<td></td>
</tr>
<tr>
<td>$2024</td>
<td>Byte</td>
<td>OutDataLastWrite</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slave address of the last written output data (set by user)</td>
<td></td>
</tr>
<tr>
<td>$2025 - $2124</td>
<td>Byte[256]</td>
<td>DPM1Melde</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Massage buffer (read only)</td>
<td></td>
</tr>
<tr>
<td>$2125 - $2224</td>
<td>Byte[256]</td>
<td>DPM1Comand</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Command buffer (written by user)</td>
<td></td>
</tr>
<tr>
<td>$2225 - $2FE9</td>
<td>Byte[3525]</td>
<td>FMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved for FMS-Services (4096-512-16-3766Bytes)</td>
<td></td>
</tr>
<tr>
<td>$2FEA</td>
<td>Byte</td>
<td>Triggerflag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(set by user)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00: free running</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x10 or 0x20: Trigger single cycle</td>
<td></td>
</tr>
<tr>
<td>$2FEB</td>
<td>Byte</td>
<td>Triggersync</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(read only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00: free running</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x11 or 0x21: cycle in work</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x10 or 0x20: cycle ready</td>
<td></td>
</tr>
<tr>
<td>$2FEC - $2FED</td>
<td>Word</td>
<td>Proficycle</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cycle counter</td>
<td></td>
</tr>
<tr>
<td>$2FEE - $2FEF</td>
<td>Word</td>
<td>Profictime</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cycle time (µs)</td>
<td></td>
</tr>
<tr>
<td>$2FF0 - $2FFF</td>
<td>Byte[16]</td>
<td>kompat</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved (only for VME-PROFI)</td>
<td></td>
</tr>
<tr>
<td>$3000</td>
<td>Byte</td>
<td>UserSema</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>User Semaphore (written by user)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$3007</td>
<td>Byte</td>
<td>Pstatus</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$3008</td>
<td>Byte</td>
<td>DPM1Csema</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication Semaphore (read only)</td>
<td></td>
</tr>
<tr>
<td>$3009 - $300E</td>
<td>Byte[6]</td>
<td>CommSema</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>$300F</td>
<td>Byte</td>
<td>Pcomand</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flag for firmware</td>
<td></td>
</tr>
</tbody>
</table>
6 Example programs

6.1 ANSI-C example program

6.1.1 DPSTART.C

/* V M E - P R O F - M   D E M O  
Example program for the startup of the VME-PROF-M in 
the VMEbus configuration mode. */

#include <stdio.h>
#include <module.h>
#include <signal.h>
#include <types.h>
#include <time.h>
#include <math.h>
#include <process.h>
#include <errno.h>

#define VS_Base 0x800000

typedef unsigned char byte ;
typedef unsigned short wort ;
typedef unsigned int lwort ;

#include "VS.h"    /* Description of the VMEbus DPR structure */
/* DPR = Dualport RAM */
#include "ProfM.h"  /* Description of the PROFIBUS data structure */

/lib 4..* Prototypen ------------------------- */
extern int DpRun (byte *fp) ;
extern void DispSlave ( byte* s);
extern void Profi_Read Daten (byte Station, wort Offset, wort Anzahl, byte *InDaten ) ;

/* Anzahl = Number */
extern void Profi_Write Daten (byte Station, wort Offset, wort Anzahl, byte *OutDaten ) ;
extern short DP_StationRun ( byte a ) ;
extern short DP_StationDia ( byte a ) ;
extern byte GetDiagnose (byte Station) ;
extern byte GlobalControl (byte RemAdr, byte GroupSel, byte Comand ) ;
extern void DpStop ( void ) ;
extern void WaitMS ( int Time ) ;
/* ---------- Globale Variablen ------------------- */
VS_Typ *DP ;
byte Speicher[65*256] ; /* Memory for 1 master-/bus-record */
and 64 slave-records */
byte Wert ;
wort *IRQflag ;

extern volatile ProfiDPR_Typ * Profi ; /* Pointer to PROFIBUS area in DPR */
extern byte MBuf[256] ;

/* ----------- Interupt Service ---------------------- */
int ProfiIRQ (void)
{
    if (*IRQflag)
    {
        *IRQflag = 0 ; /* Cler IRQ flag */
        Wert = Profi->DPM1io[0] ; /* Input Byte read */
        Wert++ ; /* if Inputs and Outputs on the Slave */
                 are connected the Count demo runs */
        Profi->DPM1io[0x800] = Wert ; /* Output Byte write */
        return 1 ;
    }
    return 0 ;
}

/* --------------- Main ----------------------------- */
int main (int argc , char **argv)
{
    int err,ret;
    u_int32 st ;
    u_int16 sink ;
    process_id pid ;
    int s , count, cyc ;
    byte *p ;
    wort n;
    FILE *fp ;
    byte Buf[8] ;
    byte ln ;
    wort LastCycle ;
    byte VMEkonfig = 0 ;

    printf ("Start program for VME-PROF-M \n");
    printf ("---------------------------- \n\n");

    /* Set Pointer to VMEbus base */
p = (byte*) 0x870000 ;

    DP = (VS_Typ*) p ;    /* DPR of the VME-PROFI */
    Profi = (ProfiDPR_Typ*) (p + 0x1000) ; /* Pointer to PROFIBUS data */
    IRQflag = (wort*) (p + 0xFFF0) ; /* Pointer to IRQ Flag Register */
/* Request the accessment rights from the Operating system */
if ( (err = _os9_id(&pid,&sink,&sink,&sink)) )
    printf("Error _os9_id() %X
", err);
if ((err = _os_permit ((void*)p, 0x10000, 0x777, pid)))
    printf("Error _os_permit() %X \n", err);

/* print Board-Ident */
printf(" Manufactor-Ident ...: %c%c%c
", DP->ID1, DP->ID2, DP->ID3);
printf(" Board-Ident ........: %3d\n", DP->BoardID);
printf(" Board-Version ......: %3X\n", DP->BoardED);

/* Init of the VME-PROFI */
printf("Init VME-PROFI : \n") ;
DP->Control = 0x00 ; /* Reset VME-PROF-M */
WaitMS (100) ;
DP->Error = 1 ; /* Pre-value, will be cleared from the VME-PROF-M */
DP->Comand = 1; /* Pre-value, will be cleared from the VME-PROF-M */

DP->Control = 0x08 ; /* 08 = enable VME-Prof-M */

printf(" Wait for Local-CPU Start ...
");
n=1000;
while (DP->Comand && n-- ) WaitMS(10); /* wait until Comand cleared or timeout */

if (VMEkonfig) /* configuration via VMEbus */
{
    DP->Comand = 1; /* When the CPU starts , it clears the Comand flag (-> 0)
    With Comand = 0 the Software will startup and reads all 
    commands from the internal EEPROM (locally stored 
    configuration).
    If Comand is set to now to 1, no data will be red from the 
    EEPROM, therefore the board can be controlled through the 
    VMEbus service interface */
    for (n=0;n<10;n++) if (DP->Error) WaitMS(1000) ;
    if (DP->Error)
    { /* Timeout after 10 sec. */
        printf(" *** Error : Local CPU did not run!\n") ;
        DP->Control = 0x00 ;
        return (0) ;
    }
    WaitMS (2000) ; /* Startup time for VME-PROFI CPU */
    printf(" Start \n") ;

    /* Loading of the configuration-file into the memory */
    A configuration-file can be generated with the program DPKONFIG
    running on a PC under Windows 95 or NT. With DPKONFIG the file
    are build by reading the slaves GSD-files.
    A configuration file consists of:
    a header 8 bytes
printf ("\nRead configuration file\n") ;
/* */
fp = fopen ("AutoLoad.dp1","r") ;
if (!fp) { printf (" Autoload.dp1 - File not found\n") ;
    return errno ;
} ;
/* the first 8 bytes are not needed */
for (n=0;n<8;n++) Speicher[n] = fgetc(fp) ; /* Reading Bus-/ Master-record (fixed length)*/
p = &Speicher[0] ;
for (n=0;n<256;n++) *p++ = fgetc(fp) ;

/* now one or more slave-records (variable length) are following */
while (!feof(fp))
    { /* reading up to the length byte */
        for (n=0;n<8;n++) Buf[n] = fgetc(fp) ;
        ln = Buf[7] ;
        if (Buf[0] == 0x25) /* 25 = service code for slave-record */
            { printf ("Reading slave-record %3d - %d Byte\n",Buf[3],ln) ;
                /* copying of the first already red 8 bytes */
                for (n=0;n<8;n++) *p++ = Buf[n] ; /* Reading of the remainig record, plus one byte*/
                for (n=8;n<ln+1;n++) *p++ = fgetc(fp) ;
            } ; /* all slave records red */
        *p++ = 0 ; /* Setting of end mark */
    fclose (fp) ;

    /* Now the hole configuration data are in memory
    ---- DpStart transfers the data via the dualport RAM service interface
    to the communication processor */

    if (!DpRun (&Speicher[0])) printf (" PROFIBUS started !\n") ;
    else
        { printf (" *** Error starting PROFIBUS DP\n") ;
            return (1) ;
        } ;
}

WaitMS (2000) ; /* Waiting for startup of the slaves */

n = irqinst ( ProfiIRQ , 95, 2) ; /* Install IRQ Service for OS-9 */
DP->Vector = 95 ; /* Set VME-PROF-M Vertor Register */
DP->Control = 0x08 + 2 ; /* Enable IRQ Level 2 */
LastCycle = 0;
n = 1000;
while (n)
{
    WaitMS(10);
    n--;
    if (Profi->ProfiCycle != LastCycle)
    {
        LastCycle = Profi->ProfiCycle;
        printf ("\n CyC= %d  CyTime = %d",LastCycle,Profi->ProfiCTime);
    }
}

/* Now PROFIBUS-services can be used and I/O- data can be transferred */

/* Show stations */
printf ("\n Showing the stations \n");
DispSlave (&Speicher[0]);

/* additionally examples */
if (DP_StationDia(10))
{
    printf ("\n Diagnose from station 10 :");
    if (!GetDiagnose(10))
        for (n=0;n<MBuf[7];n++) printf (" %02X",MBuf[8+n]);
    printf ("\n");
}

while (DP_StationRun(10))
{
    Profi_ReadDaten (10,0x000,2,&Buf[0]);
    Buf[0]++;
    Profi_WriteDaten (10,0x800,2,&Buf[0]);
}

/* DpStop () ;
*/

return (0);
} /* END MAIN */
6.1.2 PROFMLC

/* -----------------------------------------------
PROFIBUS INTERFACE
Example sub programs for VME-PROF-M boards
--
*/
#include <stdio.h>
#include "DMStyp.h"
#include "Profm.h"

/* Prototypes ------------------- */
int DpRun (byte *fp) ;
void DispSlave ( byte*s);
void Profi_ReadDaten (byte Station, wort Offset, wort Anzahl, byte *InDaten ) ;
void Profi_WriteDaten (byte Station, wort Offset, wort Anzahl, byte *OutDaten ) ;
short DP_StationRun ( byte a ) ;
short DP_StationDia ( byte a ) ;
byte GetDiagnose (byte Station) ;
byte GlobalControl (byte RemAdr, byte GroupSel, byte Comand ) ;
void DpStop (void) ;

/* Globale variables ------------ */
volatile ProfiDPR_Typ * Profi ; /* Pointer to the PROFIBUS area within the DPR*/
byte KBuf[256] ; /* Buffer for commands */
byte MBuf[256] ; /* Buffer for meassages */

/*--- Sub prgrams ----------------- */

/* PutKomm transfers a service command to the PROFIBUS firmware
try again, if the buffer was not free
Return value 0 */
byte PutKomm(void)
{
byte n;
if (((Profi->DPM1Usema) & 1) != ((Profi->DPM1Csema) & 1))
/* the command buffer is not free, if both semephores are different */
    return(0);
else
    { /* Buffer is free - copying data into the buffer */
        for (n=0; n<KBuf[7]+8; n++) Profi->DPM1Command[n] = KBuf[n];
        /* set semaphore, buffer is now ready to execute */
        Profi->DPM1Usema ^= 0x01; /* switch bit */
        return(1);
    }
/*---- Read a meassage from the dualport RAM --------*/

byte GetMeld(void)
{
    byte n;
    if (((Profi->DPM1Usema) & 0x02) == ((Profi->DPM1Csema) & 0x02))
        /* both semaphores are equal -> no message available */
        return(0);
    else
        { /* Pick up a message from the DPR */
            for (n=0; n<255; n++) MBuf[n] = Profi->DPM1Melde[n];
            /* than release the mesage buffer */
            Profi->DPM1Usema ^= 0x02;
            return(1);
        }
/* DoKomm transfers a service command to the PROFIBUS firmware and waits 
for the corresponding response message 
Return: result in Mbuf 
    status as return value 0 = O.K. 
    255 = timeout*/

byte DoKomm(void)
{
    int TimeOut;
    printf("Service : %02X", KBuf[0]);
    TimeOut = 50;
    while (TimeOut)
        { /* set command */
            if (PutKomm())
                { /* command transfered */
                    TimeOut = 50; printf (":");
                    while (TimeOut)
                        { /* waiting for the response */
                            if (GetMeld())
                                { /* message available */
                                    return(1);
                                }
                            else
                                { /* buffer is free - copying data into the buffer */
                                    for (n=0; n<KBuf[7]+8; n++) Profi->DPM1Command[n] = KBuf[n];
                                    /* set semaphore, buffer is now ready to execute */
                                    Profi->DPM1Usema ^= 0x01; /* switch bit */
                                    return(1);
                                }
                        }
                } /* command transfered */
        } /* set command */
} /* DoKomm */
if (MBuf[0] == ((KBuf[0] & 0x3F) | 0xC0))
{
    /* message is our response -> ready */
    return(MBuf[2]);
}
else /* another status message -> not used */
    printf("*" );
}
else /* wait for message */
    WaitMS (200);
    TimeOut--; 
}
if (TimeOut == 0)
{
    printf(" - Time Out - ");
    return (255);
}

/* sets the PROFIBUS-DP operating mode (=Betriebsart) for the Master */
byte SetBetriebsart (byte Art)
{
    KBuf[0] = 0x27;
    KBuf[3] = Art;
    KBuf[7] = 4;
    return (DoKomm());
}

/* Stop communication processor */
void DpStop (void)
{
    if (SetBetriebsart (0x80)) printf(" Clear FAIL\n"); /* Clear */
    else printf(" Clear O.K.\n");
    WaitMS (200);
    if (SetBetriebsart (0x40)) printf(" Stop FAIL\n"); /* Stop */
    else printf(" Stop O.K.\n");
    WaitMS (200);
    Profi->Pcommand = 0; /* Stop firmware */
}
/* Profi_ReadDaten copies the input data of a slave station from the DPR into the memory (pointer *InDaten)
The station address must be passed to ensure data consistency.
Offset is the station's Inputoffset within the DPR
Anzahl is the input data length / bytes */

void Profi_ReadDaten (byte Station, wort Offset, wort Anzahl, byte *InDaten )
{
    wort n ;
    byte old, lastStation, diff ;

    Loop:
    old = Profi->InDataState ;
    for (n=0;n<Anzahl;n++) InDaten[n] = Profi->DPM1io[Offset+n] ;
    lastStation = Profi->InDataLastWrite ;
    diff = Profi->InDataState - old ;
    if (old & 1)
        /* ungerade */
        if (diff > 1) goto Loop;
        if (Station == lastStation) goto Loop ;
    }
    else
        /* gerade */
        if (diff > 2) goto Loop ;
        if ((diff == 1) && (Station == lastStation)) goto Loop ;
    }

/* Profi_WriteDaten copies the output data of a slave station from the memory (pointer *OutDaten) into the DPR.
The station address must be passed to ensure data consistency.
Offset is the station's Outputoffset within the DPR
Anzahl is the output data length / bytes */

void Profi_WriteDaten (byte Station, wort Offset, wort Anzahl, byte *OutDaten )
{
    wort n ;
    Profi->OutDataState ++ ;
    Profi->OutDataLastWrite = Station ;
    for (n=0;n<Anzahl;n++) Profi->DPM1io[Offset+n] = OutDaten[n] ;
    Profi->OutDataState ++ ;
}

/* DP_StationRun shows the status of the slave station
The station address must be passed.
Return value: 1 if the slave is present at the bus and data are
transferred (slave is running)
0 not running        */

short DP_StationRun ( byte a )
{
    byte n , b ;
    n = a >> 3 ;
    b = 1 << (a & 7) ;
    if (Profi->DPMLive[n] & b) return 1 ;
    else return 0 ;
}

/* DP_StationDia shows the if diagnostic data are available for this slave
   The station address must be passed.
   Return value: 1 diagnostic data available
   0 no diagnostic data        */

short DP_StationDia ( byte a )
{
    byte n , b ;
    n = a >> 3 ;
    b = 1 << (a & 7) ;
    if (Profi->DPMDiag[n] & b) return 1 ;
    else return 0 ;
}

/* With GetDiagnose a diagnostic message of a slave can be red from
   the DPR.
   Return value: 0, the diagnostic data can be red starting at
   byte 8 from Mbuf. MBuf[7] is the length of the data.        */
byte GetDiagnose (byte Station)
{
    KBuf[0] = 0x21 ;
    KBuf[3] = Station ;
    KBuf[7] = 4 ;
    MBuf[2] = 0 ;
    return (DoKomm()) ;
}

/* With function GlobalControl the PROFIBUS commands SYNC and FREEZE
   can be sent to one or more slaves.
   RemAdr is the slave address 1..126 or 127 for all slave’s
   GroupSel masks that command to all slaves within the selected group (0 = all)
   Command has the following values 0x08 Freeze   - to freeze the inputs
                                    0x04 Unfreeze - inputs normal
                                    0x20 Sync     - to freeze outputs
                                    0x10 Unsync   - outputs normal
   Return value: 0 = O.K.                       */
byte GlobalControl (byte RemAdr, byte GroupSel, byte Comand )
{
    KBuf[0] = 0x2E ;
    KBuf[1] = RemAdr ;
    KBuf[3] = Comand ;
    KBuf[7] = 5 ;
    MBuf[2] = 0 ;
    return (DoKomm()) ;
}

/* DispSlave reads the configuratin data from memory and shows all configured slaves with the Username, Runflag and Diagnosticflag. The pointer to the configuration data in memmory must be passed. */
void DispSlave ( byte*s)
{
    byte ln,a,p,c,u,n ;
    if (*s == 0x24) s = s + 0x100 ; /* skip the bus-/ master-record */
    while (*s == 0x25) /* slave record */
      {
        ln = s[7] + 1 ;
        a = s[3] ;
        p = s[25] - 9 ;
        c = s[34+p] - 2 ;
        u = 48 + p + c ;
        printf (" %3d : ",a) ;
        for (n=0;n<20;n++) printf("%c",s[u+n]) ;
        if (DP_StationRun(a)) printf(" Run ");
        else printf(" Fail ");
        if (DP_StationDia(a)) printf(" ----");
        else printf(" Diag");
        printf ("\n") ;
        s = s + ln ;
      }
/* --DpRun ----------------------------- 
 starts the PROFIBUS-DP Master with configuration data from memory. The initialization must be finished:
    => VME-PROFI: board enabled
        set to remote configuration
        cleared EEPROM data
    => IPE, PCI-40, PCP-DP: Firmware loaded
    Profi pointer initialized to the DPR */
Access rights from the operating system

```c
int DpRun (byte *fp)
{
byte Status, ln ;
int n ;

printf ("\nPROFIBUS-DP Startup\n");

Status = SetBetriebsart (0x00) ;
if (!Status) printf (" Offline O.K. \n") ;
   else printf (" Offline FAIL \n") ;
WaitMS (300) ;
if (Status) goto Stop ;

if (*fp == 0x24)
{
   for (n=0;n<256;n++) { KBuf[n] = *fp++ ; } ;
   Status = DoKomm() ;
   if (!Status) printf ("Load bus- / master-record O.K. \n") ;
      else printf (" FAIL \n") ;
}
else
{
   printf("Missing bus- / master record \n") ;
   Status = 1 ;
}
WaitMS (300) ;
if (Status) goto Stop ;

while (*fp == 0x25)
{
   KBuf[0] = *fp ;
   ln = fp[7] ;
   if (KBuf[0] == 0x25)
   {
      for (n=0;n<ln+1;n++) { KBuf[n] = *fp++ ; } ;
      printf ("Load slave record station : %d", KBuf[3]) ;
      Status = DoKomm() ;
      if (!Status) printf (" O.K.\n") ;
         else printf (" FAIL\n") ;
      if (Status) goto Stop ;
   }
   Status = SetBetriebsart (0x40) ;
   if (!Status) printf (" Stop O.K. \n") ;
      else printf (" Stop FAIL \n") ;
   if (Status) goto Stop ;

Status = SetBetriebsart (0x80) ;
```
if (!Status) printf(" Clear  O.K. \n") ;
    else printf(" Clear  FAIL \n") ;
if (Status) goto Stop ;

Status = SetBetriebsart (0xC0) ;
if (!Status) printf(" Operate  O.K.\n ") ;
    else printf(" Operate  FAIL\n ") ;
if (Status) goto Stop ;

    Profi->Pcomand = 'R' ;
return (0) ;

Stop:
    Profi->Pcomand = 0 ;
return (Status) ;

} /* END */
1.1.1 PROFMLH

/* Profibus DPR ---- Version for VME-PROF-M-------------- */

typedef unsigned char byte; /* 8 Bit */
typedef unsigned short wort; /* 16 bit */
typedef unsigned int lwort; /* 32 Bit */

typedef struct {
    byte DPM1io[4096]; /* 4 KByte IO-data of the Slave´s */
    byte DPM1Live[16]; /* Live flags 1 bit per Slave */
    byte DPM1Diag[16]; /* Diag flags 1 bit per Slave */
    byte RemoteAccessFlag; /* Enable for class 2 master */
    byte InDataState; /* Flag for input consistency, read only */
    byte InDataLastWrite; /* Last updated input, read only */
    byte OutDataState; /* Flag for output consistency */
    byte OutDataLastWrite; /* Last updated output station */
    byte DPM1Melde[256]; /* Buffer for messages, read only */
    byte DPM1Comand[256]; /* Buffer for commands, write only */
    byte FMS[4096-512-16-37-6]; /* Reserved for FMS services */
    byte TriggerFlag; /*0: freilaufend, 0x10 o. 0x20 :Trigger Single */
    byte TriggerSync; /*0: freilaufend, 0x11 o. 0x21 Single in work, 0x10 o. 0x20 single done */
    wort ProfiCycle;
    wort ProfiCTime;
    byte kompat[16]; /* 16 dummy bytes, only for compatibility with older firmware versions of the VME-PROFI */
    byte DPM1Usema; /* User semafore */
    byte UserSema[6]; /* Reserved */
    byte Pstatus; /* Reserved */
    byte DPM1Csema; /* Communication CPU semafore, read only */
    byte CommSema[6]; /* Reserved */
    byte Pcommand; /* Flag for firmware */
} ProfiDPR_Typ;

typedef struct {
    byte dumy0; byte ID1;
    byte IRQSET; byte ID2;
    byte IRQEN; byte ID3;
    byte dumy3; byte BoardID;
    byte dumy4; byte BoardED;
    byte dumy5; byte Mail;
    byte dumy6; byte Vector;
    byte dumy7; byte Control;
    lwort LokalVec[252];
    byte Comand;
    byte Error;
    wort Prm[6];
    wort Live;
    byte ReadBuf[256];
    byte WriteBuf[256];
    wort DataBuf[128];
} VS_Typ;
6.2 Example for Berghof VME-S5 CPU

The VME-PROF-M is independent from the used operating system of the VMEbus Main CPU, because no Profibus software must be loaded. Only the VME-PROF-M must be enabled by writing a $08 into the control register.

The following example shows the setup for the S5 compatible VMEbus CPU from Berghof:

1. Create the following file:

   0: KC = 'MASKXX';
   3: KH = 8009; Grundeinstellungen 9 Worte
   4: KH = 1001; Slot1 Arbitermode ein
   5: KH = 2000; VMEzugriffe im NonPrivileg Mode
   6: KH = 3000; Serdat nicht verwenden
   7: KH = 4000; keine Userdef. Adressmodifier
   8: KH = 5000; Semaphoren gesperrt
   9: KH = 6000; Sysreset set Stop
  10: KH = 7000; keine Rueckwand PG-Schnittstelle
  11: KH = 8000; SysFail nicht beachten
  12: KH = 9000; Urloeschen erlaubt
  13: KH = 8101; Interruptverhalten 1 Wort
  14: KH = 1000; gesperrt
  15: KH = 8210; VME-Address-Tab 16 Worte
  16: KH = 5000; P-Bereich ab Eingangbyte 0
  17: KH = 448F; VME-Standard 0x8F0000
  18: KH = 1000; Offset 0x1000
  19: KH = 0180; Schrittweite 1, Anzahl 128 Byte
  20: KH = 7000; p-Bereich ab Ausgangsbyte 0
  21: KH = 448F; VME-Standard 0x8F0000
  22: KH = 1800; Offset 0x1800
  23: KH = 0180; Schrittweite 1, Anzahl 128 Byte
  24: KH = 71FF; q-Bereich Ausgangsbyte 255
  25: KH = 448F; VME-Standard 0x8F0000
  26: KH = 000F; VMEProfi Enable Register
  27: KH = 0101; Schrittweite 1, Anzahl 1 Byte
  28: KH = 51E0; Q-Bereich Eingangbyte 224
  29: KH = 448F; VME-Standard 0x8F0000
  30: KH = 2000; Stations Status
  31: KH = 0110; Schrittweite 1, Anzahl 16 Byte
  32: KH = EEEE; Ende

2. Add the following 2 command in the organisation Baustein 20 (OB20)

   :L KH 0008
   :T QB 255

3. At startup of the VME-S5 the two commands in OB20 are executed which enables the VME-PROF-M. Now the decentralized I/O can be accessed through the PLC P/p range.
6.3 Example for an Adept system

The following example shows, how the VME-PROF-M could be integrated in an ADEPT system:

```
PROGRAM a.profi()
; Test Program to integrate DMS Profi Board into AdeptMV Controller
; IMPORTANT: Before you are able to enable the robot power you have to run
; this program in a task other than 0. Only if the DMS board is enable SYSFAIL
; is deasserted
    CALL pr.init(); Initialize addresses
    CALL pr.def.dio(); Map Adept Signals to DMS Board
    CALL pr.check.board(); Check if DMS board is present
    CALL pr.enable(); Enable DMS board
.END

.PROGRAM pr.check.board()
; Reads the ID section of the DMS board. If it does not contain specific
; information at a certain address it is assumed the board is not available.
    $id = $CHR(IOGETB(pr.id1.adr,1))
    $id = $id+$CHR(IOGETB(pr.id2.adr,1))
    $id = $id+$CHR(IOGETB(pr.id3.adr,1))
    pr.id = IOGETB(pr.id4.adr,1)
    IF ($id <> "DMS") OR (pr.id <> ^HB) THEN
        TYPE "No DMS board found"
        HALT
    ELSE
        TYPE "DMS VME-Profi found at ", /H6, pr.base.adr
    END
.END

.PROGRAM pr.def.dio()
; Map V+ input and output signals to DMS VME Profi board
    LOCAL io.offset, io.sig
    ; Map input signals do DMS board
    io.offset = 0 ; Start at offset 0
    FOR io.sig = 1017 TO 1505 STEP 8 ; Map the input signals
        DEF.DIO io.sig = pr.in.adr+io.offset, 1 ; Map 8 signals
        io.offset = io.offset+1 ; Increase VME address by 1 Byte
    END
    ; Map output signals do DMS board
    io.offset = 0 ; Start at offset 0
    FOR io.sig = 17 TO 505 STEP 8 ; Map the output signals
        DEF.DIO io.sig = pr.out.adr+io.offset, 1 ; Map 8 signals
        io.offset = io.offset+1 ; Increase VME address by 1 Byte
    END
```

DEF.DIO io.sig = pr.out.adr+io.offset, 1; Map 8 signals
; starting at signal
; io.sig to VME address
io.offset = io.offset+1
; Increase VME address by 1 Byte

END
RETURN

.END
.PROGRAM pr.enable()
; Enable DMS VME Profi
  INPUTb pr.control.adr, 1 = ^H8    ; Write $08 to control register
RETURN
.END

.PROGRAM pr.init()
; Initialize addresses for communication with DMS VME Profi board
  pr.base.adr = ^H130000       ; VME Profi base address
;
  S21 = 1
  S20 = 0
;
  ; Attention ! V+ allows access only up to address $7FFFFF so A23 must be
  ; closed ('0'). Valid addresses are therfore $x30000 with 1<=x<=7.
  ; If AdeptVision is installed inside an AdeptMV cotroller the valid addresses
  ; are in the range of $0 - $3FFFFF
  ; If a Dual Vision System is used there is no space left for 3rd party boards
  ; at all.
  pr.id1.adr = pr.base.adr+^H1     ; Contains "D"
  pr.id2.adr = pr.base.adr+^H3     ; Contains "M"
  pr.id3.adr = pr.base.adr+^H5     ; Contains "S"
  pr.id4.adr = pr.base.adr+^H7     ; Contains $0B
  pr.control.adr = pr.base.adr+^HF ; $08 -> Enable Profibus
  pr.alive.adr = pr.base.adr+^H40E ; Board Alive section
  pr.in.adr = pr.base.adr+^H1000   ; Digital Input section
  pr.out.adr = pr.base.adr+^H1800  ; Digital Output section
RETURN
.END