SIMATIC

S7-Integration von DPV1-Slaves

Anwenderbeschreibung Datum : 02/2002

SIEMENS

SIMATIC

S7-Integration von DPV1-Slaves

Version: 1.0
Datum: 02/2002

Haftungsausschluß
Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenden Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so daß wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in der Druckschrift werden jedoch regelmäßig überprüft. Notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Technische Änderungen vorbehalten.

Inhaltsverzeichnis

1 Allgemeines

Für komplexere Geräte wurden in /1/ azyklische Dienste und Alarme definiert. Auf Slaveseite bietet Siemens jeweils Firmware für den SPC3 (DPSE, ohne Alarme) bzw. den DPC31 (V1SL) an, die dem Anwender diese Dienste auf einfachste Weise zugänglich macht.

Die heutigen DP-Master, CPUs der Familie S7-400 und die CPU 318-2 jeweils mit integrierter DP-Schnittstelle (ab Firmware-Version 3.0) und der CP 443-5 mit der Bestellnummer 6GK7 443-5DX03-0XE0 unterstützen die DPV1-Master-Funktionalität.

DP-Slaves, die im Hardware-Katalog von Step 7 unter ihrem Familiennamen zu finden sind, sind im Info-Text als DPV1-Slaves zu erkennen - wenn dies mit dem zugehörigen Schlüsselwort in der GSD-Datei entsprechend eingetragen ist.

DP-Slaves, die in STEP 7 über GSD-Dateien installiert werden, unterstützen ab GSD_Revision=3 die DPV1-Funktionalität. Diese Funktionalität ist in der FW entsprechend auszuprogrammieren.

Die Verwendung von DPV1 ist ab STEP 7 V5.1, Servicepack 2 möglich.

Mit den Diensten "RDREC" (Datensatz aus einem DP-Slave lesen mit dem SFB 52) und "WRREC" (Datensatz in einen DP-Slave schreiben mit dem SFB 53) bzw. "RALRM" (Alarm von einem DP-Slave empfangen mit dem SFB 54) kann eine azyklisch Kommunikation mit DPV1-Slaves an S7-Systemen durchgeführt werden.

2 Übersicht zum Zusammenspiel von Step 7, S7-CPU und DPV1-Slave

Hier erfolgt eine Übersicht, wie die Projektierung und die azyklischen Kommunikations- und Alarm-Aufrufe durchgeführt werden.

2.1 Projektierung eines DPV1-Slaves über GSD-Datei in HW-Config

Die GSD-Datei muß konform sein bzgl. der PROFIBUS Guideline - Order No. 2.122 "Specification for PROFIBUS Device Description and Device Integration - Volume 1: GSD V3.1".

In der GSD-Datei muß nach GSD_Revision=3 beschrieben sein, d.h. es müssen u.a. die 3 DPV1-Statusbytes in den User_Prm_Data eingetragen sein. Das DPV1-Bit muß gesetzt sein, die Alarme müssen je nach Unterstützung enabled sein.
Die genaue Bedeutung der DPV1-Statusbytes kann /1/ entnommen werden.

In /3/ sind spezielle Schlüsselworte für DPV1 definiert, die in der GSD-Datei zu beschreiben sind.
In der für unsere Beispiel-Projektierung verwendeten GSD-Datei sind die folgenden, auf DP-Erweiterungen bezogene Schlüsselworte für Slaves, eingetragen:

DPV1_Slave = 1

C1_Read_Write_supp = 1
C2_Read_Write_supp = 1
C1_Max_Data_Len = 240
C2_Max_Data_Len = 240
C1_Response_Timeout = 1
C2_Response_Timeout = 1
C2_Max_Count_Channels = 2
Max_Initiate_PDU_Length = 64

Diagnostic_Alarm_supp = 1
Process_Alarm_supp = 1
Update_Alarm_supp = 1
Extra_Alarm_SAP_supp = 1

Alarm_Sequence_Mode_Count = 32
Alarm_Type_Mode_supp = 1

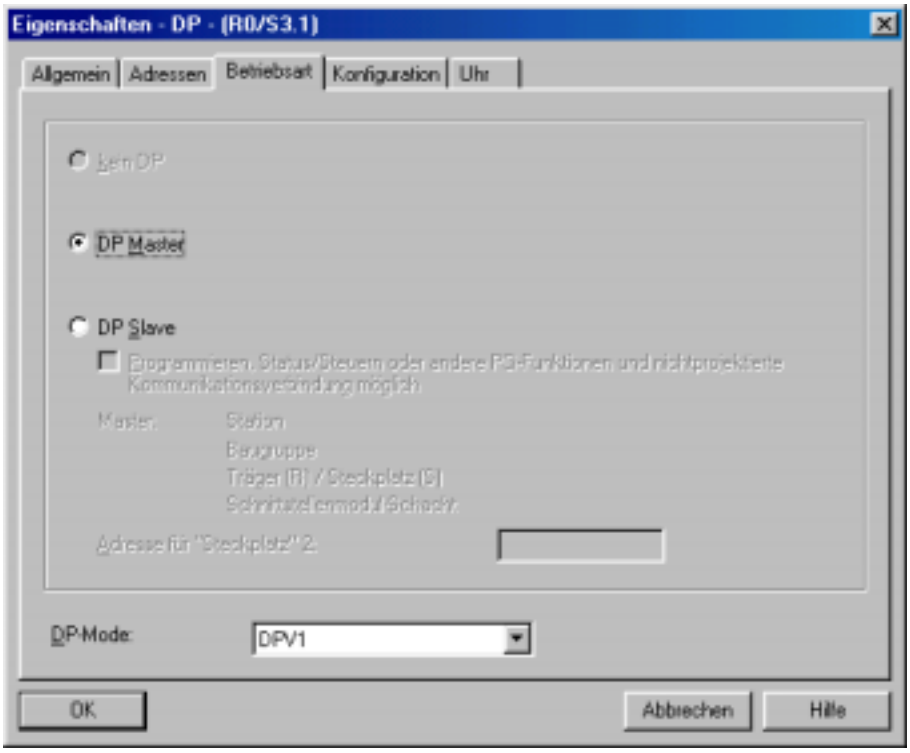
WD_Base_1ms_supp = 1

Die fett markierten Schlüsselworte müssen für den Betrieb von azyklischen Diensten immer definiert sein.
In diesem Beispiel werden von den möglichen 7 Alarmtypen nur der Diagnose-, Prozeß-, Update- und Extra-Alarm unterstützt.

Die S7-CPU´s unterstützen den Type_Mode.

Beispiel: Die Projektierung wird durchgeführt mit einem DPV1-Master "CPU 414-3 DP" und einer Simulationsbaugruppe "SM 423 SIM" mit 4 Byte E/A und einem DPV1-Slave "DPV1 Sample" basierend auf einer Eigenentwicklung namens "DPC31 Test-Board".

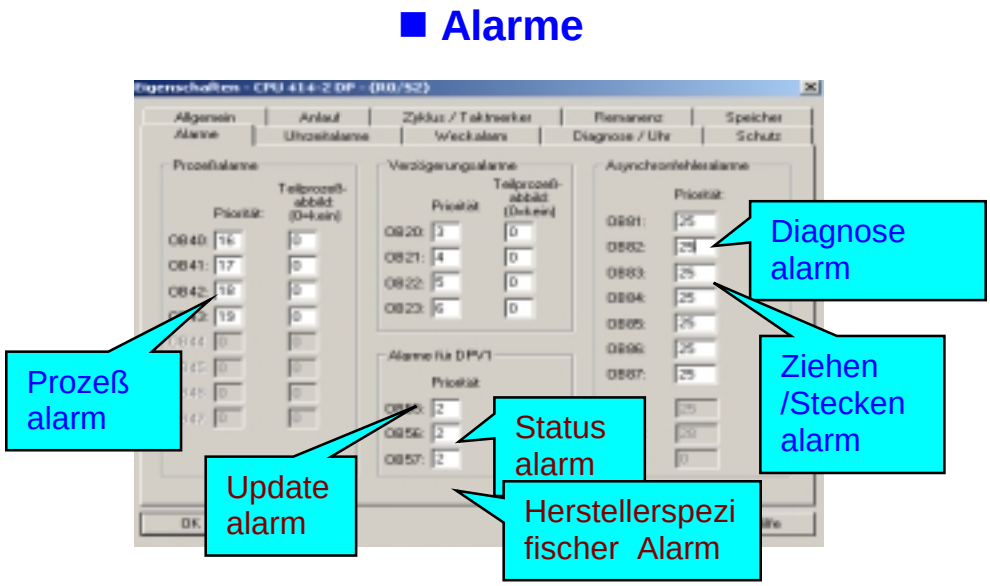
Der Master wird mit der Einstellung DP-Mode "DPV1" projektiert.



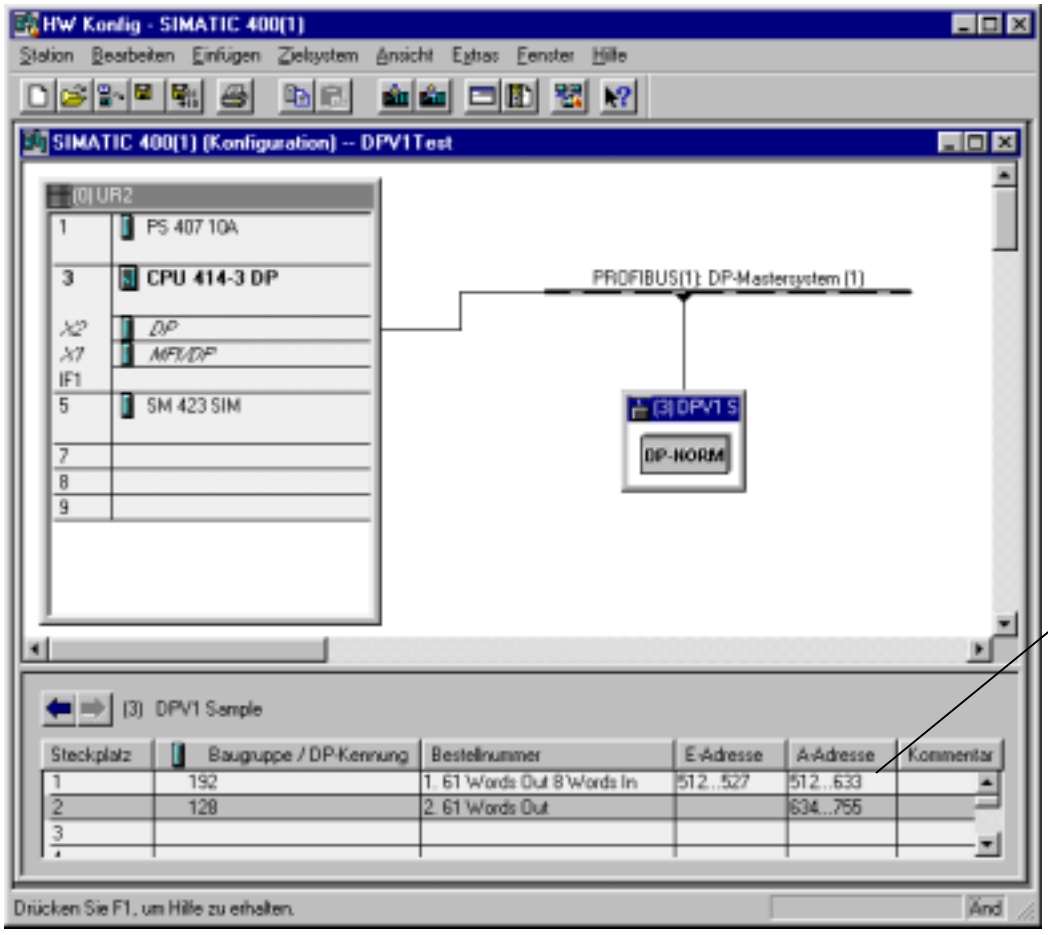
Im DPV1-Modus kann die volle DPV1-Funktionalität genutzt werden.

Die Automatisierungs-komponenten und Slaves, die kein DPV1 unterstützen, können wie gewohnt weiterhin benutzt werden, lediglich ohne Unterstützung der erweiterten Funktionen von DPV1.

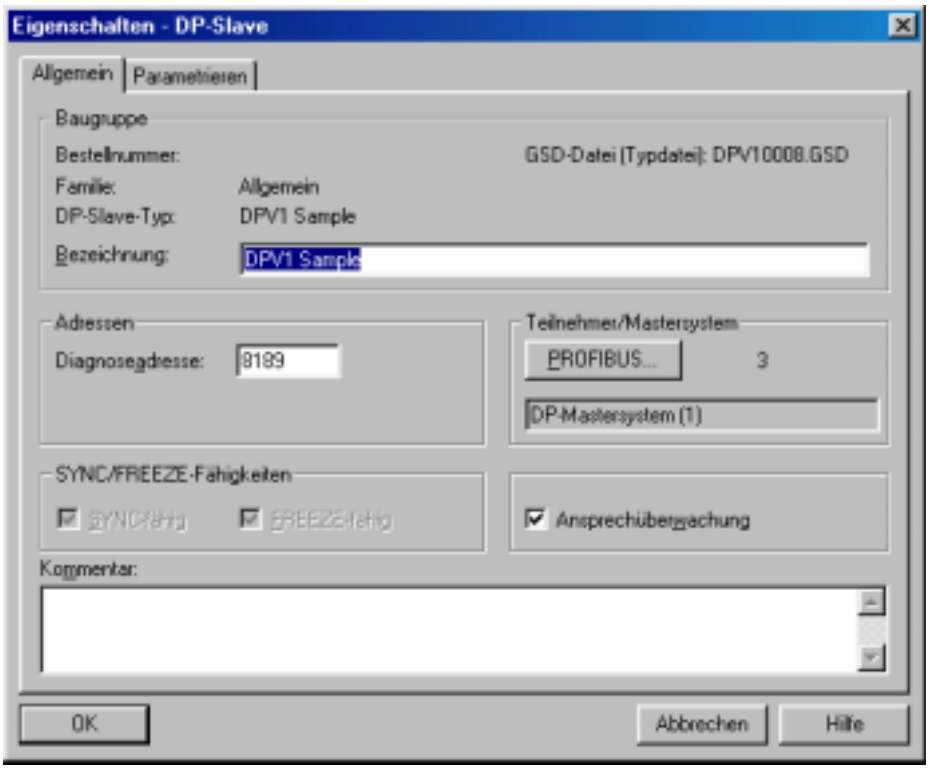
Bei den Eigenschaften der Master-CPU können im Register „Alarme“ die Prioritäten der Alarm-OB´s vorgegeben werden. Nicht alle CPUs können alle in S7 verfügbaren OBs bearbeiten. Welche OBs zur Verfügung stehen, kann den Datenblättern zur jeweiligen CPU entnommen werden.



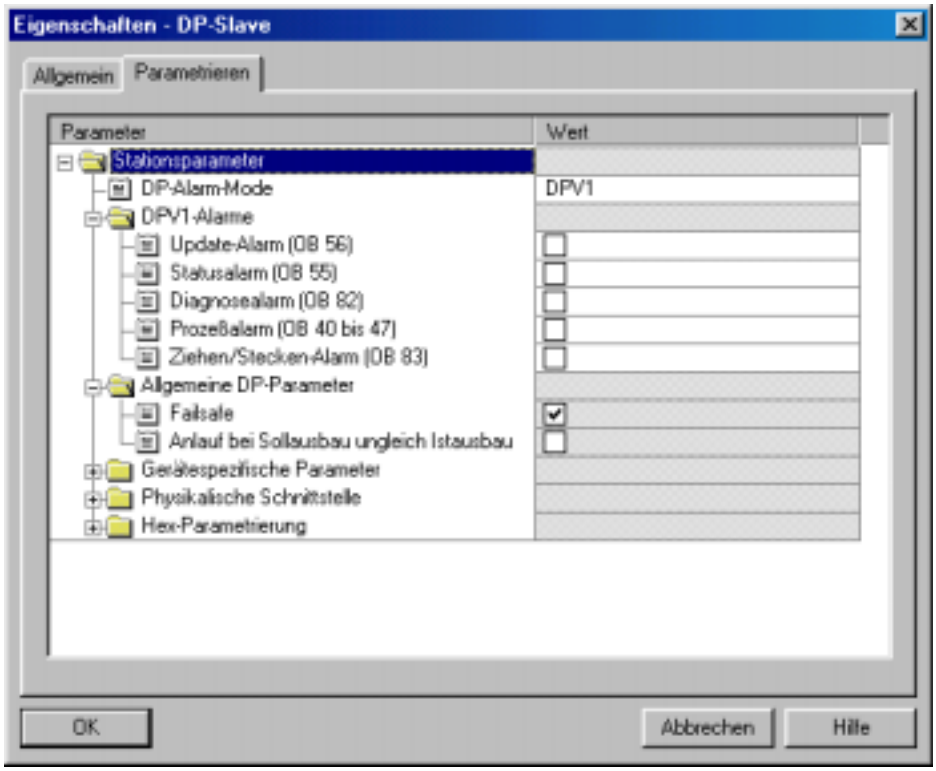
Bei der Slavekonfiguration werden den einzelnen Modulen jeweils logische E/A-Adressen im Adressraum der CPU zugewiesen. Diese logischen Adressen werden in der S7 zur Adressierung des Slaves und dessen Modulen bzw. der Datensätze der Module verwendet.

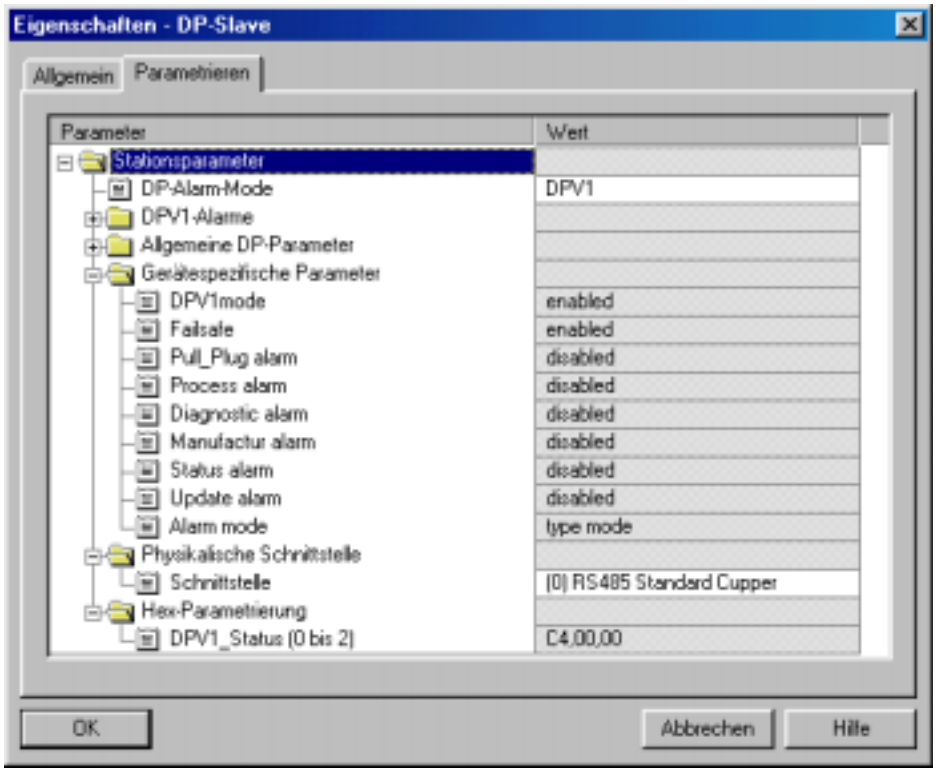


Die Adressierung des Komplettgeräts (Slot 0) kann mittels der Diagnoseadresse (hier in unserem Beispiel 8189_{DEZ} entspricht $1FFD_{HEX}$, sh. auch Bsp.-Programme Step7) erfolgen.



Im Register „Parametrieren“ können die Einstellungen zu den DPV1-Parameter-Definitionen noch hinsichtlich der unterstützten Alarm-OB´s angepasst werden.





2.2 "RDREC" Aufruf (SFB 52, Datensatz lesen) in Step 7

Mit dem SFB 52 ”RDREC” wird ein Datensatz von der adressierten Baugruppe gelesen.

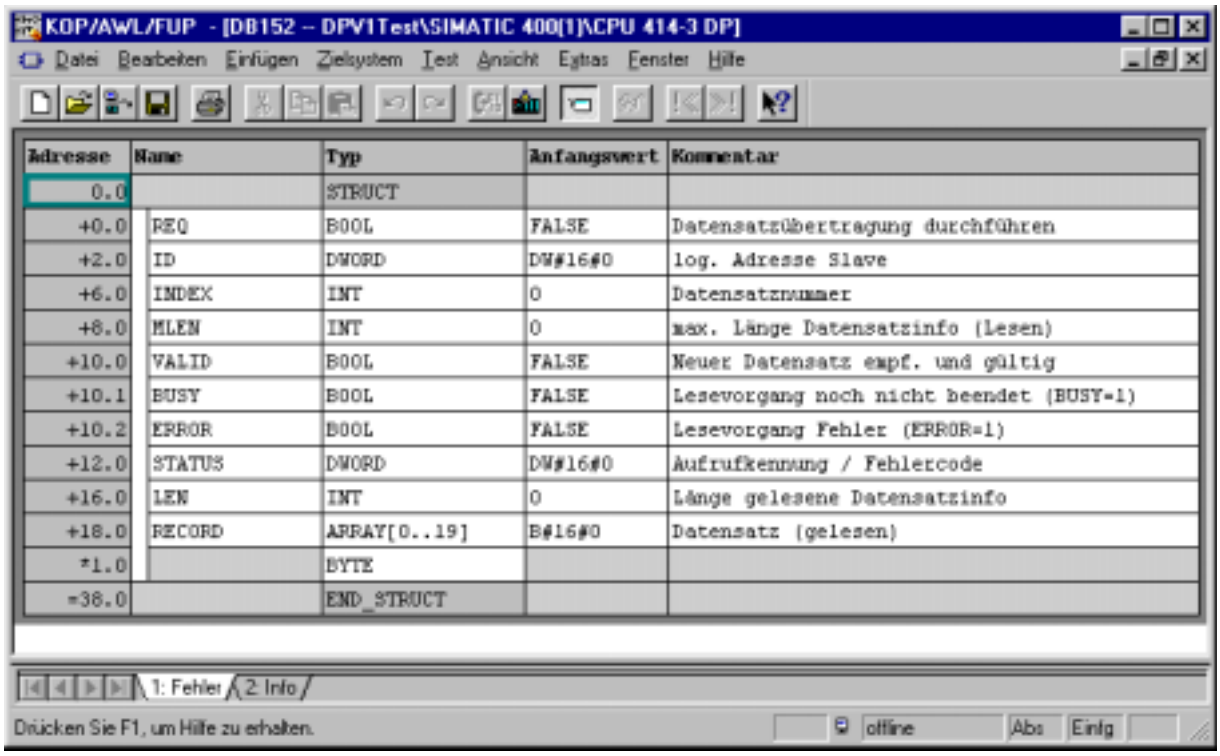
Eine ausführlichen Beschreibung des Aufrufs, die Versorgung der Parameter und die Fehlercodes ist in /2/ enthalten.

Beispiel:

```
U     E      0.0                  // Anstoß Datensatz lesen
UN    M     10.1                  // Hilfsmerker
=     M     10.2                  // Flankenmerker

U     E      0.0
=     M     10.1

CALL  "RDREC" , DB52
 REQ   :=M10.2
 ID    :=DW#16#1FFD
 INDEX :=7
 MLEN  :=4
 VALID :="RDREC_DB".VALID
 BUSY  :="RDREC_DB".BUSY
 ERROR :="RDREC_DB".ERROR
 STATUS:="RDREC_DB".STATUS
 LEN   :="RDREC_DB".LEN
 RECORD:="RDREC_DB".RECORD
```

Dieser SFB-Aufruf liest den Datensatz 7 mit der Länge von 4 Byte aus dem Slave mit der Diagnoseadresse 0x1FFD und legt die empfangenen Daten im "RDREC_DB" (DB 152) ab.

Struktur des "RDREC_DB" :



2.3 "WRREC" Aufruf (SFB 53, Datensatz schreiben) in Step 7

Mit dem SFB 53 "WRREC" wird ein Datensatz in die adressierten Baugruppe geschrieben.

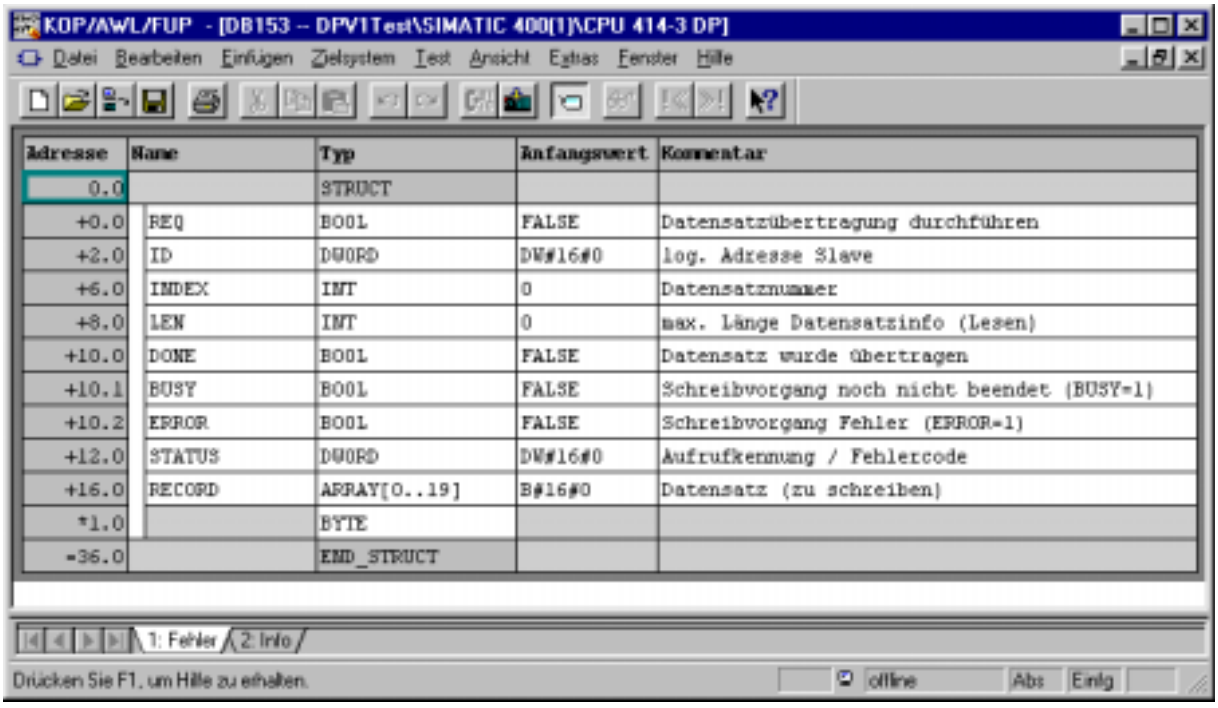
Eine ausführlichen Beschreibung des Aufrufs, die Versorgung der Parameter und die Fehlercodes ist in /2/ enthalten.

Beispiel:

```
U     E      0.1                  // Anstoß Datensatz schreiben
UN    M     11.1                  // Hilfsmerker
=     M     11.2                  // Flankenmerker

U     E      0.1
=     M     11.1

L     EB     3                    // EB 3 verwenden als
T     "WRREC_DB".RECORD[0]        // erstes Byte von Datensatz

CALL  "WRREC" , DB53
 REQ   :=M11.2
 ID    :=DW#16#1FFD
 INDEX :=5
 LEN   :=3
 DONE  :="WRREC_DB".DONE
 BUSY  :="WRREC_DB".BUSY
 ERROR :="WRREC_DB".ERROR
 STATUS:="WRREC_DB".STATUS
 RECORD:="WRREC_DB".RECORD
```

Dieser SFB-Aufruf schreibt den Datensatz 5 mit der Länge von 3 Byte, der im "WRREC_DB" (DB 153) abgelegt ist, an den Slave mit der Diagnoseadresse 0x1FFD.

Struktur des "RDREC_DB" :



2.4 "RALRM" Aufruf (SFB 54, Alarm empfangen) in Step 7

Mit dem SFB 54 ”RALRM” wird ein Alarm samt der zugehörigen Information (Startinformation des aufgerufenen OB und Informationen aus der Alarmquelle) von einer Komponente eines DP-Slaves gelesen und an seinen Ausgangsparametern zur Verfügung gestellt.

Der SFB 54 darf nur aus dem Alarm-OB aufgerufen werden, den das Betriebssystem der CPU aufgrund des zu untersuchenden Alarms aus der Peripherie gestartet hat.

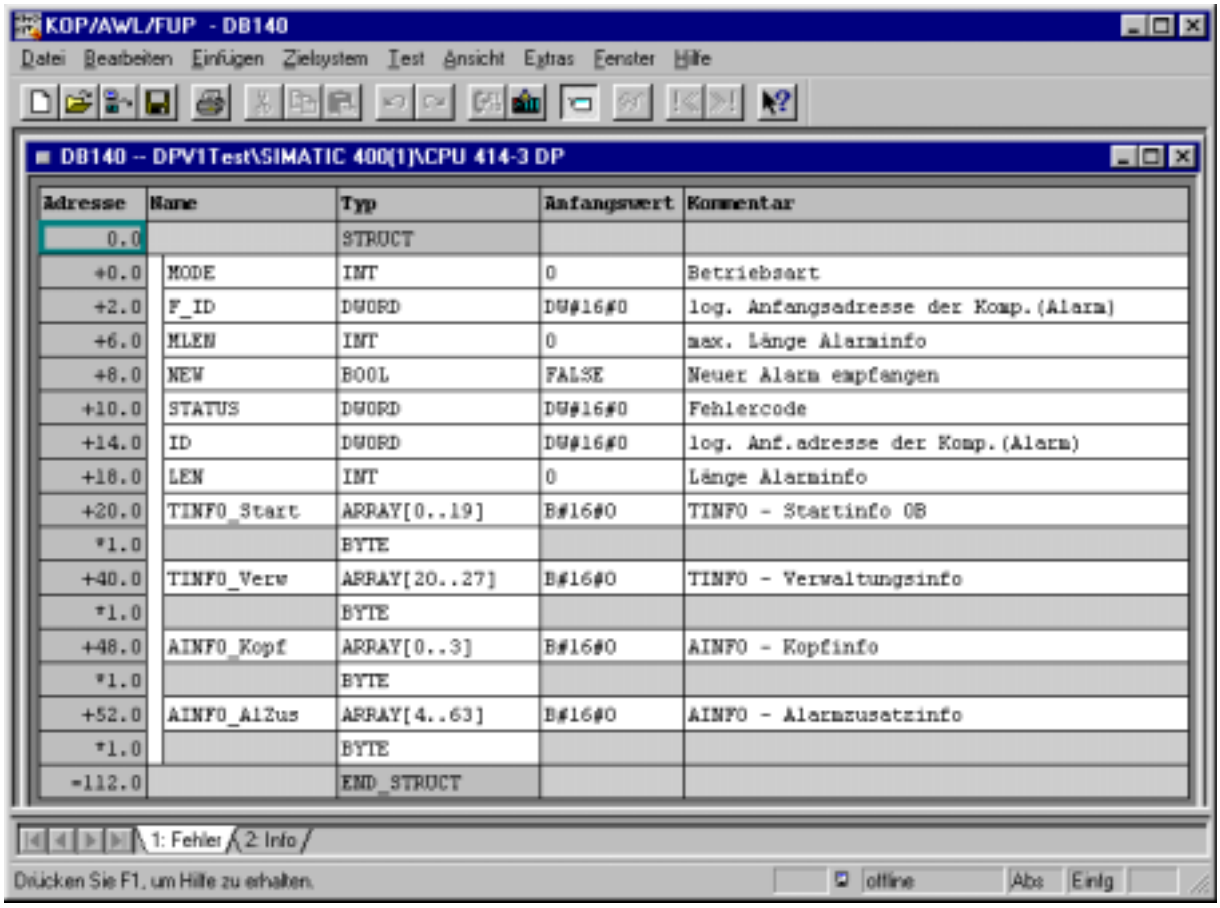
Eine ausführlichen Beschreibung des Aufrufs, die Versorgung der Parameter und die Fehlercodes ist in /2/ enthalten.

Beispiel OB40:

```
CALL  "RALRM" , DB54
 MODE  :=1
 F_ID  :=DW#16#1FFD
 MLEN  :=244
 NEW   :="PRALRM_DB".NEW
 STATUS:="PRALRM_DB".STATUS
 ID    :="PRALRM_DB".ID
 LEN   :="PRALRM_DB".LEN
 TINFO :="PRALRM_DB".TINFO_Start
 AINFO :="PRALRM_DB".AINFO_Kopf
```

Dieser SFB-Aufruf empfängt einen Prozeßalarm, der vom OB 40 (Prozeßalarm 0) gemeldet wurde, und legt die Alarminformationen im "PRALRM_DB" (DB 154) ab.

Struktur des "RDREC_DB" :



2.5 "DS_READ" Callbackfunktion im Slave

```
/*------------------------------------------------------------------------------*/
/* c1 ds_read pdu received                                                      */
/*------------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_c1_read_ds(V1SL_LL_DS_READ_PTR ds_read_ptr )
{
  /* todo: do c1 ds_read handling here */
  /* example begin */
  usr_read_ds(ds_read_ptr);
  /* example end */
  v1sl_c1_read_ds_done();
}


/*------------------------------------------------------------------------------*/
/* generic ds_read function                                                     */
/*------------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_read_ds( V1SL_LL_DS_READ_PTR ds_read_ptr )
{
  DS_TEMP V1SL_LL_DATA_ATTR *p_ds;

  /* in this example only index 2 is allowed */
  switch( ds_read_ptr->req.index )
  {
  case 2:
    if(ds_read_ptr->req.user_data_len > sizeof(DS_TEMP))
    {
      ds_read_ptr->req.user_data_len = sizeof(DS_TEMP);
    }

    p_ds = ((DS_TEMP V1SL_LL_DATA_ATTR *)&(ds_read_ptr->res.user_data));
    p_ds->upper_phys_limit = SWAP_WORD(usr_data.ds_temp.upper_phys_limit);
    p_ds->lower_phys_limit = SWAP_WORD(usr_data.ds_temp.lower_phys_limit);
    p_ds->upper_alarm_level = SWAP_WORD(usr_data.ds_temp.upper_alarm_level);
    p_ds->upper_warning_level = SWAP_WORD(usr_data.ds_temp.upper_warning_level);
    p_ds->lower_warning_level = SWAP_WORD(usr_data.ds_temp.lower_warning_level);
    p_ds->lower_alarm_level = SWAP_WORD(usr_data.ds_temp.lower_alarm_level);
    p_ds->time_factor = SWAP_WORD(usr_data.ds_temp.time_factor);
    p_ds->enable_alarms = usr_data.ds_temp.enable_alarms;
    break;

  default:
    ds_read_ptr->nrs.function_number |= V1SL_FN_ERROR_BASE;
    ds_read_ptr->nrs.error_decode = V1SL_ED_DPV1;
    ds_read_ptr->nrs.error_code_1 = V1SL_EC1_DPV1_CLASS_ACCESS |
                                    V1SL_EC1_ACC_CODE_INDEX_INVALID;
    /* error_code2 should not be used */
    ds_read_ptr->nrs.error_code_2 = 0;
    break;
  }
}
```

2.6 "DS_WRITE" Callbackfunktion im Slave

```
/*------------------------------------------------------------------------------*/
/* c1 ds_write pdu received                                                     */
/*------------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_c1_write_ds(V1SL_LL_DS_WRITE_PTR ds_write_ptr )
{
  /* todo: do c1 ds_write handling here */
  /* example begin */
  usr_write_ds(ds_write_ptr, FALSE);
  /* example end */
  v1sl_c1_write_ds_done();
}


/*------------------------------------------------------------------------------*/
/* generic ds_write function                                                    */
/*------------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_write_ds( V1SL_LL_DS_WRITE_PTR ds_write_ptr,
                                     Boolean c2_access )
{
  DS_TEMP V1SL_LL_DATA_ATTR *p_ds;
  V1SL_IFA_ALARM_PTR          p_alarm;
  /* index 0,1,241..255 are reserved */
  /* in this example only index 2 is allowed */
  switch( ds_write_ptr->req.index )
  {
  case 2:
    if(ds_write_ptr->req.user_data_len == sizeof(DS_TEMP))
    {
      p_ds = ((DS_TEMP V1SL_LL_DATA_ATTR *)&(ds_write_ptr->req.user_data));
      usr_data.ds_temp.upper_phys_limit = SWAP_WORD(p_ds->upper_phys_limit);
      usr_data.ds_temp.lower_phys_limit = SWAP_WORD(p_ds->lower_phys_limit);
      usr_data.ds_temp.upper_alarm_level = SWAP_WORD(p_ds->upper_alarm_level);
      usr_data.ds_temp.upper_warning_level = SWAP_WORD(p_ds->upper_warning_level);
      usr_data.ds_temp.lower_warning_level = SWAP_WORD(p_ds->lower_warning_level);
      usr_data.ds_temp.lower_alarm_level = SWAP_WORD(p_ds->lower_alarm_level);
      usr_data.ds_temp.time_factor = SWAP_WORD(p_ds->time_factor);
      usr_data.ds_temp.enable_alarms = p_ds->enable_alarms;
    }
    else
    {
      ds_write_ptr->nrs.function_number |= V1SL_FN_ERROR_BASE;
      ds_write_ptr->nrs.error_decode = V1SL_ED_DPV1;
      ds_write_ptr->nrs.error_code_1 = V1SL_EC1_DPV1_CLASS_APPLICATION |
                                       V1SL_EC1_ACC_CODE_WRITE_LEN;
      ds_write_ptr->nrs.error_code_2 = 0;
    }
    break;

  default:
    ds_write_ptr->nrs.function_number |= V1SL_FN_ERROR_BASE;
    ds_write_ptr->nrs.error_decode = V1SL_ED_DPV1;
    ds_write_ptr->nrs.error_code_1 = V1SL_EC1_DPV1_CLASS_ACCESS |
                                     V1SL_EC1_ACC_CODE_INDEX_INVALID;
    /* error_code2 should not be used */
    ds_write_ptr->nrs.error_code_2 = 0;
    break;
  }
}
```

2.7 Alarmbehandlung im Slave

```
/*-------------------------------------------------------------------------*/
/* alarm-ack pdu received                                                  */
/*-------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_al_alarm_ack(V1SL_IFA_ALARM_PTR alarm_ptr )
{
   /* alarm_ack from master received */
   /* todo: do your alarm-acknowledge handling here */
   /* example begin */
   usr_free_alarm_buffer(alarm_ptr);
   /* example end */
   return;
}


/*-------------------------------------------------------------------------*/
/* alarm statemachine state changed                                        */
/*-------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_al_state_report(Unsigned8 alarm_type_bit_field, Unsigned8
sequence_depth )
{
   sequence_depth = sequence_depth;

   if( alarm_type_bit_field == 0 )
   {
      /* alarmstatemachine stopped, remove pending alarms */
      v1sl_al_withdraw_alarm(V1SL_ALARM_TYPE_ALL_VALUE, V1SL_SEQUENCE_NUMBER_ALL );
      usr_data.al.state = USR_AL_STATE_CLOSED;

   }
   else
   {
      /* enable setting alarms */
      usr_data.al.state = USR_AL_STATE_OPEN;
   }
   return;
}


/*-------------------------------------------------------------------------*/
/* user_init_alarm_buffers function                                        */
/*-------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_init_alarm_buffers(void)
{
   Unsigned8   i;

   for(i=0;i<=V1SL_SEQUENCE_NUMBER_MAX;i++)
   {
      usr_data.al.buffer_used[i] = FALSE;
   }
}
```

```
/*----------------------------------------------------------------------------*/
/* usr_alloc_alarm_buffer function                                            */
/*----------------------------------------------------------------------------*/

V1SL_IFA_ALARM_PTR USR_IFA_CODE_ATTR usr_alloc_alarm_buffer(void)
{
   V1SL_IFA_ALARM_PTR ptr = NULL;
   Unsigned8 i;

   for(i=0;i<=V1SL_SEQUENCE_NUMBER_MAX;i++)
   {
      if(usr_data.al.buffer_used[i] == FALSE)
      {
         ptr = &(usr_data.al.alarm_buffers[i]);
         usr_data.al.buffer_used[i] = TRUE;
         ptr->specifier=i<<3;
         ptr->user_data_len = 1;
         ptr->user_data_ptr = &(usr_data.al.alarm_buffer_user_data[i]);
         break;
      }
   }
   return ptr;
}

void USR_IFA_CODE_ATTR usr_free_alarm_buffer(V1SL_IFA_ALARM_PTR ptr)
{
   usr_data.al.buffer_used[(ptr->specifier)>>3] = FALSE;
}


/*----------------------------------------------------------------------------*/
/* usr_send_state_alarm function                                              */
/*----------------------------------------------------------------------------*/

void USR_IFA_CODE_ATTR usr_send_state_alarm(TEMP_STATE temp_state, TEMP_STATE
old_temp_state)
{
   V1SL_IFA_ALARM_PTR  ptr;

   ptr=usr_alloc_alarm_buffer();
   if(ptr)
   {
      ptr->slot_number = 0;
      *(ptr->user_data_ptr) = temp_state;

      if((temp_state == TS_HIGH_ALARM) || (temp_state == TS_LOW_ALARM))
      {
         /* coming process alarm */
         ptr->specifier |= V1SL_SPEC_ALARM_SPEC_SLOT_ERR_VALUE;
         ptr->alarm_type = V1SL_ALARM_TYPE_PROC;
      }
      else if((temp_state == TS_NORMAL))
      {
         /* going diag alarm */
         ptr->specifier |= V1SL_SPEC_ALARM_SPEC_NO_ERR_VALUE;
         ptr->alarm_type = V1SL_ALARM_TYPE_DIAG;
      }
```

```
      else if(old_temp_state == TS_NORMAL)
      {
         /* coming diag alarm */
         ptr->specifier |= V1SL_SPEC_ALARM_SPEC_SLOT_ERR_VALUE;
         ptr->alarm_type = V1SL_ALARM_TYPE_DIAG;
      }
      else
      {
         /* going process alarm, pending diag alarm */
         ptr->specifier |= V1SL_SPEC_ALARM_SPEC_SLOT_ERR_VALUE
            +V1SL_SPEC_ALARM_SPEC_NO_ERR_VALUE;
         ptr->alarm_type = V1SL_ALARM_TYPE_PROC;
      }

      if(V1SL_OK_ASYNC != v1sl_al_set_alarm(ptr))
      {
         /* alarm could not be send */
         usr_free_alarm_buffer(ptr);
      }
   }
   else
   {
      /* no buffer available */
   }
}



/* example generate update-alarm */

            p_alarm = usr_alloc_alarm_buffer();
            if(p_alarm)
            {
               p_alarm->alarm_type = V1SL_ALARM_TYPE_UPDT;
               p_alarm->slot_number = 0;
               if(V1SL_OK_ASYNC != v1sl_al_set_alarm(p_alarm))
               {
                  /* alarm could not be send */
                  usr_free_alarm_buffer(p_alarm);
               }

            }
```

3 Randbedingungen

3.1 Slave-Adressierung entsprechend der logischen Adressierung

Bei der S7 erfolgt eine Umrechnung von logischen Adressen zu Slaveadressen.

Die Durchführung von "RDREC", "WRREC" und "RALRM" mit den Slaves erfolgt über die logische Adresse (Diagnose-, E/A-Adresse).

Die Slotadressierung für Datensatz Lesen/Schreiben erfolgt direkt über die E/A-Adresse.

4 DPV1-Unterstützende Master

Die Funktionen "RDREC", "WRREC", "RALRM" werden derzeit von folgenden Mastern (CPUs mit integrierter DP-Schnittstelle und dem CP443-5) unterstützt:

Master	Bestell-Nummer	FW-Stand ab
CPU 412-1	6ES7 412-1XF03-0AB0	V 3.0
CPU 412-2	6ES7 412-2XG00-0AB0	
CPU 414-2	6ES7 414-2XG03-0AB0	
CPU 414-3	6ES7 414-3XJ00-0AB0	
CPU 416-2	6ES7 416-2XK02-0AB0	
CPU 416-3	6ES7 416-3XL00-0AB0	
CPU 417-4	6ES7 417-4XL00-0AB0	
CPU 612-2	6ES7 612-2QH00-0AB4	
CPU 616-2	6ES7 616-2QL00-0AB4	
CPU 414-4H	6ES7 414-4HJ00-0AB0	
CPU 417-4H	6ES7 417-4HL01-0AB0	
CPU 318-2	6ES7 318-2AJ00-0AB0	
CP 443-5	6GK7 443-5DX03-0XE0	

5 Beispiel-Programme

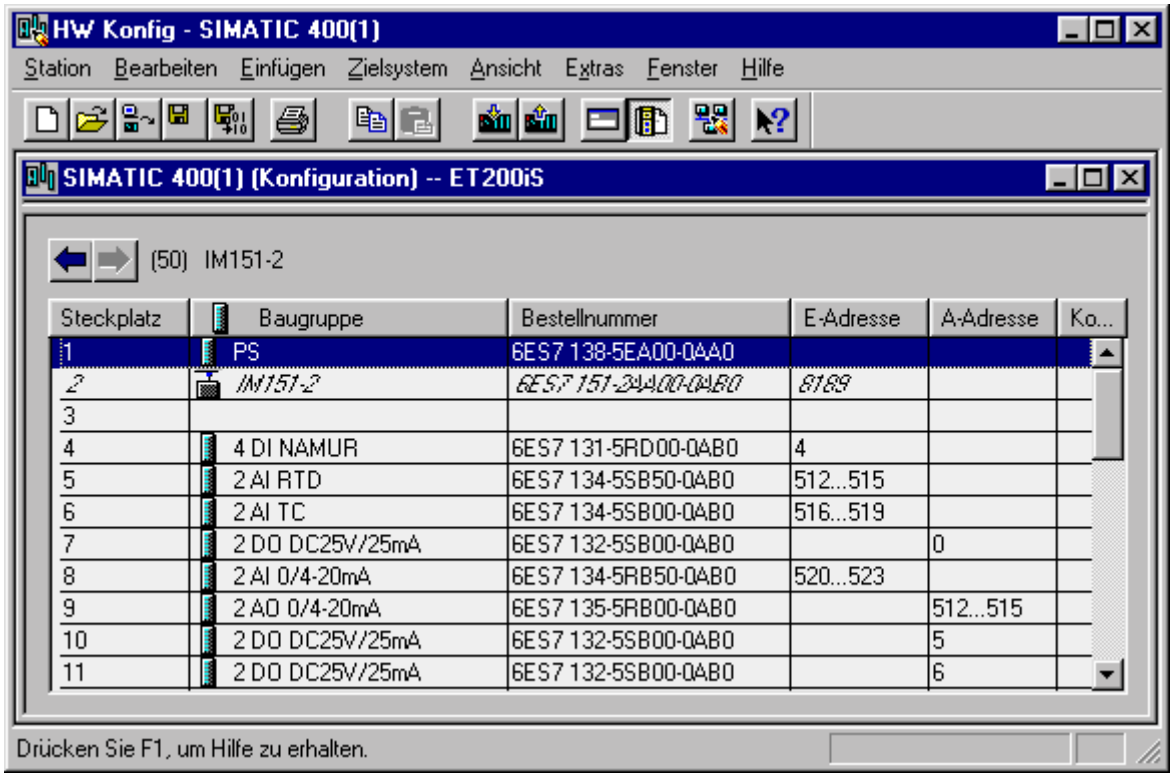
Sie haben die Möglichkeit, zwei Beispielprogamme, durchgeführt mit einem DPV1-Master “CPU 414-3 DP“, vom Internet zu laden. Im Master-Rack ist zusätzlich eine Simulationsbaugruppe “SM 423 SIM“ mit 4 Byte E/A projektiert.

Beispiel 1:
Als Slave ist ein “DPV1 Sample“ projektiert, die Baugruppe ist das “DPC31 TestBoard“ und stammt aus dem Entwicklungspaket DPV1PA von Siemens. Die Konfiguration besteht aus 2 Modulen,

1. “61 Words Out 8 Words In“
2. “61 Words Out“

Als Baudrate ist 1,5 Mbit/s eingestellt.

Beispiel 2:
Als Slave ist eine ET200iS (IM151-2) projektiert. Für die Projektierung ist zusätzlich Simatic PDM V 5.1 installiert.
Die Konfiguration ist wie folgt:



Als Baudrate ist 1,5 Mbit/s eingestellt.

Beide Beispielprogramme (STEP7-Projekt, DPC31-Code, GSD-Datei) und diese Anwenderbeschreibung sind von den Internetseiten des Customer Support im Bereich “Automation & Drives“ zu laden.

Sie sind zu finden beim “Produkt Support“ unter
Produktinformationen->Automatisierungssysteme->Dezentrale Peripherie->Technologie Komponenten ->Allgemeines
im Register “Handbücher / BA“.

Es gibt auch einen Direktzugriff, Adresse “http://www.ad.siemens.de/csi/pb-doc“

6 Anhang

6.1 Anschriften

PROFIBUS Nutzer Organisation

PNO
Geschäftsstelle
Dr. Wenzel
Haid- und Neu- Straße 7
76131 Karlsruhe
Tel.: (0721) 9658-590

Technische Ansprechpartner im Schnittstellencenter in Deutschland

Siemens AG
A&D SE RD 73
Hr. W. Götz

Briefadresse:
Postfach 2355
90713 Fürth

Hausadresse
Würzburgerstr.121
90766 Fürth

Tel.: (0911) 750 - 2074
Fax: (0911) 750 - 2100

Email: Werner.Goetz@siemens.com

6.2 Weiterführende Literatur

/1/ PROFIBUS-DP Extensions to EN50170 (DPV1) Version 2.0, April 1998

/2/ Systemsoftware für S7-300/400 System- und Standardfunktionen, Referenzhandbuch, A5E00069891-03

/3/ PROFIBUS Guideline
Specification for PROFIBUS Device Description and Device Integration
Volume 1: GSD V 4.1, July 2001

Siemens AG
Bereich Automatisierungstechnik
Kombinationstechnik
Postfach 2355, D-90713 Fürth

SIEMENS Aktiengesellschaft

Printed in the Fed. Rep. of Germany