SIEMENS

SIMATIC NET

SPC3 und DPS2

Softwarebeschreibung

(Siemens PROFIBUS Controller
nach IEC 61158)

Version: 1.1

Datum: 04/09

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.

Versionskennungen

Version	Datum	Änderung
V 1.0	09.04.03	Entfernen der HW- spezifischen Teile. Diese sind in der SPC3 Hardwarebeschreibung zu finden. Optimierung der max TSDR Zeiten.
V1.0	27.04.09	Ansprechpartner

Inhaltsverzeichnis

1 Einführung

Für den einfachen und schnellen digitalen Datenaustausch zwischen Automatisierungsgeräten bietet Siemens seinen Anwendern mehrere ASIC's an, die auf Grundlage der PROFIBUS Norm (IEC 61158) den Datenverkehr zwischen den einzelnen Automatisierungsstationen unterstützen bzw. komplett abwickeln.

Zur Unterstützung von intelligenten Slavelösungen, d. h. Implementierungen mit einen Mikroprozessor stehen folgende ASICs zur Verfügung:
Der **ASPC2** hat bereits auch viele Teile der Schicht 2 integriert, benötigt aber Unterstützung durch einen Prozessor. Dieser ASIC unterstützt Baudraten bis zu 12 MBaud, ist aber mit seiner Komplexität eher für Masteranwendungen konzipiert.

Der **SPC3** entlastet durch die Integration des kompletten PROFIBUS-DP Protokolls den Prozessor eines intelligenten PROFIBUS Slave entscheidend und kann am Bus mit einer Baudrate bis 12 MBaud betrieben werden.

Im Bereich der Automatisierung gibt es aber auch einfache Geräte wie Schalter, Thermoelemente etc. für deren Zustandserfassung kein Mikroprozessor notwendig ist.
Zur preisgünstigen Anpassung dieser Geräte stehen zwei weiterer ASICs mit der Bezeichnung **SPM2** (Siemens Profibus Multiplexer, Version 2) und **LSPM2** (Lean Siemens PROFIBUS Multiplexer) zur Verfügung. Diese Bausteine arbeiten als DP-Slaves im Bussystem und arbeiten mit Baudraten bis zu 12 MBaud. Ein Master spricht sie über Schicht 2 des 7 Schichtenmodells an. Nachdem diese Bausteine ein fehlerfreies Telegramm empfangen haben, generieren sie selbständig die angeforderten Antworttelegramme.
Der LSPM2 besitzt die gleichen Funktionen wie der SPM2, jedoch mit verringerter Anzahl an I/O- und Diagnose-Ports.

Die vorliegende Dokumentation beinhaltet dieSoftwarebeschreibung DPS2.

Softwarebeschreibung:
Dieses Kapitel erklärt die seitens Siemens im Sourcecode angebotene Software zum Ansteuern des ASIC´s. Falls Sie die Software selbst schreiben möchten, können Sie das anhand der Informationen in Kapitel 1 durchführen.

Beispielprogramm:
Unter dieser Rubrik ist ein Beispielprogramm unter Anwendung der von Siemens angebotenen Software im C- Sourcecode abgedruckt, damit Sie einen Überblick über die Einfachheit der zu integrierenden Software bekommen.

2 Funktionsübersicht

2.1 Allgemeines

Der SPC3 ermöglicht einen kostenoptimierten Aufbau von intelligenten PROFIBUS-DP Slave Applikationen. Das Prozessorinterface unterstützt folgende Prozessoren:

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

Im SPC3 ist das komplette DP-Slave-Protokoll integriert.

Als Schnittstelle zwischen dem SPC3 und der Software/Anwendung dient das **integrierte 1,5kB-RAM**. Der gesamte Speicher ist in 192 Segmente zu je 8 Byte unterteilt. Die Adressierung vom Anwender erfolgt direkt und vom internen Mikrosequenzer (MS) mittels eines sogenannten Base-Pointers, der wahlfrei auf ein Segment im Speicher positioniert werden kann. Alle Puffer müssen daher immer auf einen Segmentanfang gelegt werden.
Soll der SPC3 eine DP-Kommunikation fahren, so richtet er alle DP-SAPs selbsttätig ein. Dem User werden die verschiedenen Telegramminformationen in getrennten Datenpuffern (z.B.: Parametrierdaten, Konfigurierdaten etc.) bereitgestellt. Für die Daten-Kommunikation werden sowohl für die Output- als auch für die Input-Daten 3 Wechselpuffer bereitgestellt. Dabei steht der Kommunikation immer ein Wechselpuffer zur Verfügung und damit kann kein Ressourcen-Problem auftauchen. Zur optimalen Unterstützung der Diagnose besitzt der SPC3 zwei Diagnose-Wechselpuffer, in die der User die aktuellen Diagnosedaten einträgt. Dabei ist immer ein Diagnosepuffer dem SPC3 zugeordnet.

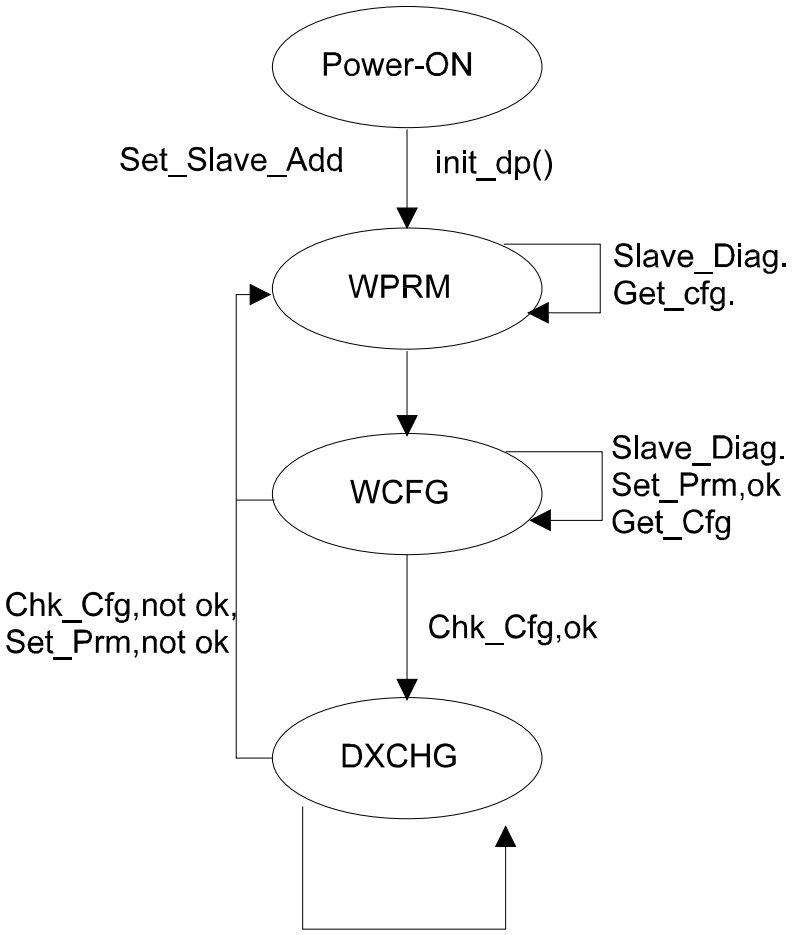
Auf den kommenden Seiten finden sie alle Informationen inkl. Eines Beispielprogramms, die sie für den Einsatz der DPS2 Software für den zyklischen Datenaustausch benötigen. Für den zyklischen und azyklischen Datenaustausch bietet Ihnen die Siemens AG zusätzlich die DPSE Software an.

3 PROFIBUS Entwicklungspaket für den SPC3

Mit dem Kauf eines Entwicklungspaketes haben sie auch die Lizenz für die DPS2 Software erworben. Im Entwicklungspaket ist eine PROFIBUS DP Masterbaugruppe (IM 180) als ISA- Baugruppe, die Busverdrahtung, sowie 2 Slave Baugruppen enthalten (Best Nr. 6ES7 195-3BA00-0YA0).

3.1 Der PROFIBUS DP Slave Zustandsautomat

Zum besseren Verständnis sei im folgenden eine Kurzbeschreibung des Zustandsautomaten eines DP-Slaves wiedergegeben.



DataExchange,ok
Rd_inp,
Rd_Outp,
Kommandos (Sync, Freeze...)
Slave_Diag,
Chk_Cfg,ok,
Set_Prm,ok
Get_Cfg

Für das Verständnis des Firmwareablaufs ist der prinzipielle Ablauf dieser Statemachine hilfreich. Die Details sind der Norm zu entnehmen.

3.1.1 Power_On

Nur im Zustand Power On wird ein Set_Slave_Address Telegramm akzeptiert.

3.1.2 Wait_Prm

Der Slave erwartet nach dem Hochlauf ein Parametertelegramm. Alle anderen Telegrammarten werden abgewehrt bzw. nicht bearbeitet. Der Datenaustausch ist noch nicht möglich.
Im Parametertelegramm sind mindestens die von der Norm festgelegten Informationen, wie z. B. PNO-Identnummer, Sync-Freeze-Fähigkeit usw. hinterlegt. Des weiteren sind userspezifische Parameterdaten möglich. Die Bedeutung dieser Daten legt allein die Anwendung fest. In der Projektierung der Masteranschaltung sind z. B. bestimmte Bits gesetzt um einen gewünschten Meßbereich anzuzeigen. Die Firmware stellt diese userspezifischen Daten dem Anwendungsprogramm zur Verfügung, welches die Daten auswertet und akzeptiert oder auch abwehren kann (z.B. der gewünschte Meßbereich ist nicht einstellbar und damit ein sinnvoller Betrieb nicht möglich).

3.1.3 Wait_Cfg

Das Konfigurationstelegramm legt die Anzahl der Ein- und Ausgangsbytes fest. Der Master teilt dem Slave mit wieviele Bytes E/A übertragen werden. Der Anwendung wird die angeforderte Konfiguration zur Überprüfung mitgeteilt. Diese Überprüfung ergibt dann entweder eine richtige, falsche oder eine anpaßbare Konfiguration. Wenn sich der Slave an die gewünschte Konfiguration anpassen will, muß eine neue Nutzdatenlänge aus den Konfigurationsbytes berechnet werden (z. B. 4 Byte E vordefiniert, 3 Byte nur genutzt). Die Anwendung muß entscheiden, ob diese Anpaßbarkeit sinnvoll ist.
Zusätzlich besteht die Möglichkeit von jedem Master die Konfiguration eines beliebigen Slaves abzufragen.

3.1.4 Data_Exchange

Wenn sowohl die Parametrierung und die Konfigurierung von der Firmware und von der Anwendung als richtig akzeptiert wurde geht der Slave in den Zustand Data_Exchange über, d. h. er tauscht Nutzdaten mit dem Master aus.

3.1.5 Diagnose

Über die Diagnose teilt der Slave dem Master seinen aktuellen Zustand mit. Dieser besteht zumindest aus den in der Norm festgelegten Informationen in den ersten sechs Octets, wie z. B. Zustand der Statemachine. Diese Informationen kann der Anwender durch prozeßspezifische Informationen (Anwenderdiagnose), ergänzen (z. B. Drahtbruch).
Die Diagnose kann als Fehlermeldung und als Statusmeldung auf Initiative des Slaves hin abgesetzt werden. Dabei beeinflußt der Anwender neben drei definierten Bits auch die anwendungsspezifischen Diagnosedaten. Es darf aber auch jeder (nicht nur der zugeteilte Master) die aktuelle Diagnoseinformation abfragen.

-> Bitte beachten Sie die ausführlichere Diagnosebeschreibung im Anhang !

3.1.6 Read_Inputs, Read_Outputs

Jeder Master kann von jedem beliebigen Slave (im Zustand Data_Exchange) die aktuellen Zustände der Ein- und Ausgänge abfragen. Diese Funktion wird vom ASIC bzw. von der Firmware selbständig bearbeitet.

3.1.7 Watchdog

Mit dem Parametertelegramm erhält der Slave auch einen Watchdogwert. Wird dieser Watchdog durch den Busverkehr nicht nachgetriggert, geht die Statemachine in den „sicheren“ Zustand Wait_Prm über.

3.2 Optimierung der max TSDR Zeiten beim ASIC SPC3

In Anlagen, bei denender Isochron Mode benutzt wird kann es nützlich sein, die Antwortzeiten des Slaves zu optimieren. Die Zeit, wann ein Slave spätestens auf eine Aufforderung vom Master antworten muss idt di max TSDR Zeit. Die in der Norm angegebenen Werte werden vom SPC3 deutlich unterschritten, was zu einer Optimierung des Buszyklusses führt. Geben sie deshalb in der GSD- Datei folgende Werte ein:

Baudrate in kbit/s	Bis 187,5	500	1500	3000	6000	12000
MaxTSDR	15	15	25	50	100	200

4 DPS2

4.1 Einleitung

Der PROFIBUS DP ASIC SPC3 entlastet einen angeschlossenen Mikroprozessor fast völlig von der Bearbeitung der PROFIBUS DP Statemachine. Er hat Funktionen, die bei früheren ASICs durch die zugehörige Firmware ausgeführt werden mußten, fest im internen Mikroprogramm integriert.

Die Schnittstelle zum Anwender ist das aus der Hardwarebeschreibung ersichtliche Register bzw RAM-Interface.

Das Programmpaket DPS2 für den SPC3 entlastet den Anwender des SPC3 von hardwarenahen Registermanipulationen und Speicherberechnungen. DPS2 bietet eine komfortable „C“-Schnittstelle und insbesondere eine Unterstützung bei der Einrichtung der Pufferorganisation. Ein Übergang von DPS2 für den SPC2 auf DPS2/SPC3 ist einfach möglich, da die Aufrufe und Organisation gleich sind.

Das gesamte Programmpaket besteht aus

Modul		Funktion
userspc3.c	Hauptprogramm	Hier werden die Funktionen: Hochlauf, Ein-/Ausgabe und Diagnose bedient.
intspc3.c	Interruptmodul	Diese Modul behandelt die Funktionen: Parametrierung, Konfigurierung.
dps2spc3.c	Hilfsfunktionen	Diese Funktionen berechnen aus der gewünschten Konfiguration die Pufferorganisation.
spc3dps2.h	Makros und Definitionen	Diese Makros ermöglichen einen einfachen Zugriff des Anwenders auf die Registerstruktur des ASICs.

DPS2 benötigt als Schnittstelle zum Anwender einen Interrupt für den SPC3, den der Anwender einrichten muß. Die Funktionen, die beim Auftreten des ASIC Interrupts auszuführen sind, enthält das Programm intspc3.c .

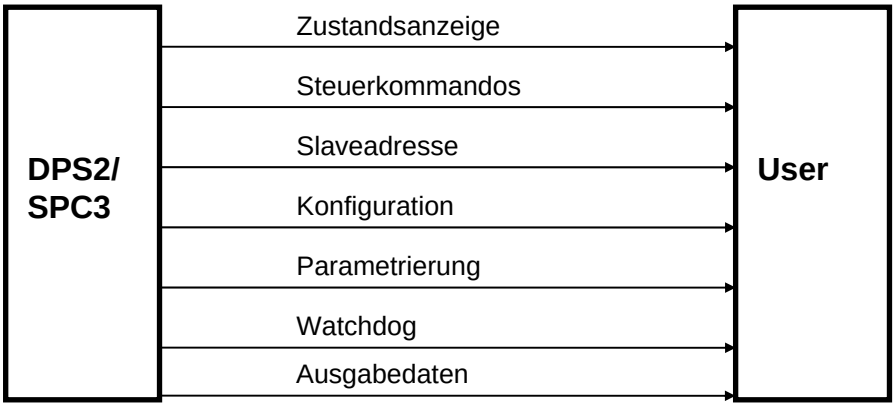
Dieser Interrupt kann vom Anwenderprogramm zeitweise gesperrt werden. Es ist auch möglich den Interrupt prinzipiell zu sperren und die entsprechenden Funktionen im Pollingverfahren abzuarbeiten.

Die Schnittstelle zwischen Anwender und der DPS2 -Firmware teilt sich auf in Funktionen,

- die von der Anwendung zur Verfügung gestellt und von DPS2 aufgerufen werden

und Funktionen,

- die von DPS2 zur Verfügung gestellt und von der Anwendung aufgerufen werden.



4.2 Initialisierung

4.2.1 Hardware

Im ersten Schritt des Hochlaufs setzt das Anwenderprogramm den ASIC SPC3 über den RESET Pin zurück, initialisiert das interne RAM des SPC3 und die Reseteinstellungen des angeschlossenen Prozessors

4.2.2 Compilereinstellungen

Das Literal SPC3_INTEL_MODE gibt die Beschaltung des entsprechenden Pins am SPC3 an, d. h. stellt die Darstellung der Wortregister im SPC3 ein.

SPC3_INTEL_MODE/_INTEL_COMP		
Übergabe	#define	Intel-Interface des SPC3 gewählt
	nicht definiert	Motorola-Interface des SPC3 gewählt
Rückgabe	------	

Prozessor	Compiler	Einstellungen	Bemerkungen
SAB 165	Boston Tasking	SPC3_INTEL_MODE	
80C32	Keil Compiler	SPC3_INTEL_MODE	Compiler stellt Wortgrößen in Motorola Format dar => Der Swapmechanismus der Makros muß aktiviert werden.

Mit #define DPS2_SPC3 wird die DPS2 Schnittstelle aktiviert.

Um die unterschiedlichen Speichermodelle bei einigen Prozessoren zu unterstützen, werden die Zugriffe auf den SPC3 mittels Zeiger mit einem Zeigerattribut SPC3_PTR_ATTR versehen. Dieses Attribut ist Compiler-abhängig.

Für den C166-Compiler muß die Adreßlage des SPC3 angegeben werden:
```
#define SPC3_NEAR        /* der SPC3 liegt im NEAR-Adreßbereich */
#define SPC3_FAR         /* der SPC3 liegt im FAR-Adreßbereich */
```

Für den 80C32-Compiler muß die Adreßlage der Userdaten angegeben werden:
```
#define SPC3_DATA_XDATA  /* die Userdaten liegen externen RAM-Adreßbereich */
#define SPC3_DATA_IDATA  /* der Userdaten liegen im interne RAM */
```

Mit dem Literal #define SPC3_NO_BASE_TYPES kann die Deklaration der Basistypen (UBYTE, BYTE, UWORD, WORD) unterdrückt werden.

4.2.3 SPC3 locatieren

Zum Anlegen des SPC3 als Struktur im Speicherbereich ist ein Typ SPC3 definiert. Compiler-abhängig muß diese Struktur auf den durch die Hardware festgelegten Adreßbereich mit dem Namen spc3 gelegt werden.

4.2.4 "Hardware"-Modus

Das Makro SPC3_SET_HW_MODE(|) ermöglicht diverse Einstellungen des SPC3.

SPC3_SET_HW_MODE(x)	Hardwareeinstellungen	
Übergabe		
	INT_POL_LOW	Der Interrupt-Ausgang ist low-aktiv.
	INT_POL_HIGH	Der Interrupt-Ausgang ist high-aktiv.
	EARLY_RDY	Ready wird um einen Takt vorgezogen.
	SYNC_SUPPORTED	Der Sync_Mode wird unterstützt.
	FREEZE_SUPPORTED	Der Freeze_Mode wird unterstützt.
	DP_MODE	Der DP_Mode ist freigegeben, alle DP_SAPs werden vom SPC3 eingerichtet. Diese Einstellung wird automatisch gesetzt.
	EOI_TIMEBASE_1u	Die Interrupt-Inaktivzeit ist mindestens 1 usec lang.
	EOI_TIMEBASE_1m	Die Interrupt-Inaktivzeit ist mindestens 1 ms lang.
	USER_TIMEBASE_1m	Der User_Time_Clock-Interrupt kommt alle 1 ms.
	USER_TIMEBASE_10m	Der User_Time_Clock-Interrupt kommt alle 10 ms.nochmal näher beschreiben!!!
	SPEC_CLEAR	Der SPC3 soll Failsave-Telegramme akzeptieren
Rückgabe	------	

Der User_Time_Clock ist ein frei für die Anwendung verfügbarer Timer. Er erzeugt einen 1 bzw. 10 ms Timertakt, der durch eine entsprechende Freigabe zu einem Interrupt führt.

4.2.5 Aktivierung der Benachrichtigungsfunktion

Mit dem Makro SPC3_SET_IND (|) werden die Benachrichtungsfunktionen bzw. die Interruptauslöser freigeschalten. Die Übergabeparameter sind immer ein UWORD. Die BYTE- (Endung _B) und BIT-Literals (Endung:_NR) sind für die effektiveren Makros bzw. zur Auswertung nach Laden in einen bitadressierbaren Bereich bestimmt.

SPC3_SET_IND(x|x..)		Benachrichtigungsfeld aktivieren
Übergabe	MAC_RESET	Der SPC3 ist, nachdem er den aktuellen Auftrag abgearbeitet hat, in den *Offline-Zustand* gekommen (durch Setzen des Bits 'Go_Offline').
	GO_LEAVE_DATA_EX	Die DP_SM ist in den Zustand 'DATA_EX' eingetreten oder hat ihn verlassen.
UWORD	BAUDRATE_DETECT	Der SPC3 hat den 'Baud_Search-Zustand' verlassen und eine Baudrate gefunden.
Darstellung	WD_DP_MODE_TIMEOUT	Im WD-Zustand 'DP_Control' ist der Watchdog-Timer abgelaufen.
	USER_TIMER_CLOCK	Die Zeitbasis des freiverfügbaren User_Timer_Clocks ist abgelaufen (1/10ms -Timertick).
	NEW_GC_COMMAND	Der SPC3 hat ein 'Global_Control-Telegramm' mit einem geänderten 'GC_Command-Byte' empfangen und dieses Byte in die RAM-Zelle 'R_GC_Command' abgelegt.
	NEW_SSA_DATA	Der SPC3 hat ein 'Set_Slave_Address-Telegramm' empfangen und die Daten im SSA-Puffer bereitgestellt.
	NEW_CFG_DATA	Der SPC3 hat ein 'Check_Cfg-Telegramm' empfangen und die Daten im Cfg-Puffer bereitgestellt.
	NEW_PRM_DATA	Der SPC3 hat ein 'Set_Param-Telegramm' empfangen und die Daten im Prm-Puffer bereitgestellt.
	DIAG_BUFFER_CHANGED	Auf Anforderung durch 'New_Diag_Cmd' hat der SPC3 die Diagnose-Puffer ausgetauscht und den alten wieder dem User zur Verfügung gestellt.
	DX_OUT	Der SPC3 hat ein 'Write_Read_Data-Telegramm' empfangen und die neuen Output-Daten im -Puffer bereitgestellt. Bei 'Power_On' bzw. bei einem 'Leave_Master' löscht der SPC3 den Inhalt desPuffers und generiert auch diesen Interrupt.
Rückgabe	------	

Beispiel:
SPC3_SET_IND(GO_LEAVE_DATA_EX | WD_DP_MODE_TIMEOUT);
/* Der Anwender wird benachrichtigt, wenn der Zustand DATA_Exchange eingetreten ist oder verlassen wird oder der Watchdogtimer abgelaufen ist */

Eine Interruptfreischaltung mit Bytegrößen könnte folgendermaßen aussehen:
SPC3_SET_IND(NEW_CFG_DATA_B | NEW_PRM_DATA_B | USER_TIMER_CLOCK_B);

4.2.6 User-Watchdog

Der User-Watchdog sorgt dafür, daß beim Ausfall des angeschlossenen Mikroprozessors der SPC3 nach einer definierten Anzahl (DPS2_SET_USER_WD_VALUE) von Datentelegrammen den Datenzyklus verläßt. Solange der Mikroprozessor nicht „abstürzt“, muß er diesen Watchdog nachtriggern (DPS2_RESET_USER_WD).

DPS2_SET_USER_WD_VALUE (x)		User Watchdog Zeit einstellen
Übergabe	UWORD	Anzahl der Datentelegramme
Rückgabe	------	

DPS2_RESET_USER_WD()		Neustart / Nachtriggern des User Watchdogs
Übergabe	------	
Rückgabe	------	

Im Worst Case können die Datentelegramme im Zeitintervall des Min_Slave Intervalls gesendet werden.
Die Anwendung kann mittels dieser Zeitangabe und der Laufdauer des eigenen Programmteiles die Anzahl der Datentelegramme festlegen.
Beispielrechnung: ($T_{laufzeit\ der\ Applikation}$ / Min_Slave Intervall) x 2 = Anzahl Datentelegramme

Min_Slave Intervall - siehe hierzu DIN E 19245 Teil3 (minimale Pollzeit des Masters von Telegrammen an den Slave).
2 - Sicherheitsfaktor

4.2.7 Stationsadresse

Beim Hochlauf liest das Anwendungsprogramm die Stationsadresse (DIL-Schalter, EEPROM, etc.) ein und übergibt diese dem ASIC. Des weiteren legt der Anwender fest, ob diese Stationsadresse über den PROFIBUS DP geändert werden kann, d. h. ein Speichermedium (z. B. serielles EEPROM) steht zur Verfügung.

SPC3_SET_STATION_ADRESS (x)		Stationsadresse setzen
Übergabe	UBYTE	Adresse
Rückgabe	------	

DPS2_SET_ADD_CHG_DISABLE()		Stationsadressänderung gesperrt
Übergabe	------	
Rückgabe	------	

DPS2_SET_ADD_CHG_ENABLE()		Stationsadressänderung erlaubt Achtung: Puffer für diesen Dienst muß der Anwender einrichten !
Übergabe	------	
Rückgabe	------	

4.2.8 Identnummer

Beim Hochlauf liest das Anwendungsprogramm die Identnummer (EPROM, Hostsystem) ein und übergibt diese dem ASIC.

DPS2_SET_IDENT_NUMBER_HIGH(x)		Identnummer
Übergabe	UBYTE	Highbyte der PNO Identnummer
Rückgabe	------	

DPS2_SET_IDENT_NUMBER_LOW(x)		Identnummer
Übergabe	UBYTE	Lowbyte der PNO Identnummer
Rückgabe	------	

Die Identnummer dient der eindeutigen Identifizierung des Slaves und wird bei jedem Diagnosetelegramm vom Slave zum Master mitgesendet. Die Identnummer muß bei der PNO (siehe Anschriftenverzeichnis) beantragt werden.

4.2.9 Antwortzeit

Falls es besondere Umstände erfordern, kann der Anwender die Antwortzeit des SPC3 im Hochlauf einstellen. Im Betrieb mit PROFIBUS DP wird diese durch das Parametriertelegramm des PROFIBUS DP Masters festgelegt.

SPC3_SET_MINTSDR(x)		MinTsdr
Übergabe	UBYTE	Antwortzeit in Bitzeiten (11-255)
Rückgabe	------	

4.2.10 Puffer-Initialisierung

Dem Makro SPC3_INIT muß der Anwender eine Struktur dps2_buf vom Typ DPS2_BUFINIT übergeben, in dem die Längen der Wechselpuffer für die verschiedenen Telegramme eintragen sind. Diese Längen bestimmen die im ASIC eingerichteten Datenpuffer und sind deshalb in der Gesamtsumme vom ASIC Speicher abhängig. DPS2_INIT überprüft die eingetragenen Maximallängen der Puffer und gibt das Prüfergebnis zurück.

```
typedef struct {
  UBYTE din_dout_buf_len;     /*Gesamtlaenge des Ein-/Ausgangspuffers, 0-488*/
  UBYTE diag_buf_len;         /*Laenge des Diagnosedatenpuffers, 6-244*/
  UBYTE prm_buf_len;          /*Laenge des Parameterpuffers, 7-244*/
  UBYTE cfg_buf_len;          /*Laenge des Konfig-Datenpuffers, 1-244*/
  UBYTE ssa_buf_len;          /*Laenge des Set-Slave-Add-Puffers, 0 bzw. 4-244*/
  } DPS2_BUFINIT;
```

Bei dem Set-Slave-Address-Puffer bewirkt die Angabe der Länge 0 die Sperre dieses Dienstes.

Bei dieser Art der Puffer-Initialisierung ist noch ein weiteres Makro zur Anpassung der Längen der Din-/Dout-Puffer notwendig, da dies die einzigen sind, die im Betrieb geändert werden dürfen (allerdings nicht über die voreingestellte Größe hinaus).

SPC3_INIT (x)	Pufferinitialisierung	
Übergabe	DPS2_BUF_INIT *dps2_bptr	Zeiger auf Werte mit der Struktur DPS2_BUFINIT, d,. h. gewünschte/erforderliche Pufferlängen
Rückgabe	DPS2_INITF_DIN_DOUT_LEN	Fehler bei der Din/Doutlänge
	DPS2_INITF_DIAG_LEN	Fehler bei der Diagnoselänge
	DPS2_INITF_PRM_LEN	Fehler bei der Parametrierdatenlänge
	DPS2_INITF_SSA_LEN	Fehler bei der Adreßdatenlänge
	SPC3_INITF_LESS_LEN	insgesamt zuviel Speicher verbraucht. Es konnten nicht alle Puffer angelegt werden.
	SPC3_INITF_NOFF	Der SPC3 befindet sich nicht im Offline-Zustand
	SPC3_INIT_OK	Pufferlänge ok, SPC3 fehlerfrei initialisiert

4.2.11 Eintrag der Sollkonfiguration

Die Funktion holt sich zunächst mit dem Makro einen Zeiger auf einen Datenblock für die Konfiguration.

DPS2_GET_READ_CFG_BUF_PTR()		Zeiger auf Konfigurationspuffer holen
Übergabe	----	
Rückgabe	UBYTE *	Zeiger auf RAM-Bereich im SPC3

In diesen Datenblock trägt der Anwender seine Konfiguration (Kennungsbytes) ein. Dabei sind die einzelnen Kennungsbytes entsprechend nachfolgender Vorschrift (siehe auch EN 50170 Vol. 2) zu bilden:

Bit

7	6	5	4	3	2	1	0

Länge der Daten 00 = 1Byte/Wort
15 = 16Byte/Worte

Ein- /Ausgabe 00 = spezielles Kennungsformat
01 = Eingabe
02 = Ausgabe
11 = Ein-Ausgabe

Länge 0 = Byte, Bytestruktur
1 = Wort

Konsistenz über 0 = Byte oder Wort
1 = gesamte Länge

So entsprechen z.B. die Kennungen 17 hex = 8 Byte Eingabe ohne Konsistenz
27 hex = 8 Byte Ausgabe ohne Konsistenz

Die speziellen Kennungsformate sind der EN 50170 Vol. 2 zu entnehmen.

Mit dem Makro DPS2_SET_READ_CFG_LEN (CFG_LEN) setzt der Anwender die Länge der eingetragenen Konfigurationsdaten.

DPS2_SET_READ_CFG_LEN (x)		Länge der Konfigurationsdaten setzen
Übergabe	UBYTE	Länge der Eintragungen im Konfigurationspuffer
Rückgabe	----	

Anschließend benutzt der Anwender die im File dps2spc3.c zur Verfügung gestellte Funktion dps2_calculate_inp_outp_len(), um aus den Kennungsbytes die Länge der Eingangs- und Ausgangsdaten zu ermitteln. Diese Funktion liefert einen Pointer auf eine Struktur vom Typ DPS2_IO_DATA_LEN zurück. Ein Null-Pointer meldet eine fehlerhafte Projektierung der Puffer (z.B. real_cfg_data_len = 0).

dps2_calculate_inp_outp_len(x,y)		Berechnung der Eingänge/Ausgänge
Übergabe	UBYTE *	Zeiger auf Konfigurationspuffer
	UWORD	Länge der Konfigurationsdaten
Rückgabe	DPS2_IO_DATA_LEN *	Zeiger auf Struktur mit den berechneten Ein-, Ausgangslängen

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

Mit dem Makro DPS2_SET_IO_DATA_LEN(ptr) initiiert der User die DPS2-Variablen inp_data_len und outp_data_len.

DPS2_SET_IO_DATA_LEN(x)		Ein-/Ausgangsdatenlänge setzen
Übergabe	DPS2_IO_DATA_LEN *	Zeiger auf Struktur mit den berechneten Ein-, Ausgangslängen
Rückgabe	UBYTE	TRUE:genügend Speicher vorhanden FALSE: Speicher nicht ausreichend

4.2.12 Holen der ersten Pufferzeiger

Vor dem ersten Eintrag seiner Eingangsdaten muß sich die Anwendung mit dem Makro DPS2_GET_DIN_BUF_PTR() einen Puffer für die Eingabedaten holen. Mit dem Makro DPS2_INPUT_UPDATE() können die Eingangsdaten vom Anwender an DPS2 übergeben werden. Die Länge der Eingaben wird nicht bei jedem Update übergeben, sie muß mit der durch DPS2_SET_IO_DATA_LEN() übergebenen Länge übereinstimmen.

Makro DPS2_GET_DIN_BUF_PTR()		ersten Eingangsdatenpuffer holen
Übergabe	-----	
Rückgabe	UBYTE *	Zeiger auf Eingangsdatenpuffer.

Vor dem ersten Eintragen von externen Diagnosedaten muß sich der Anwender mit dem Makro DPS2_GET_DIAG_BUF_PTR() einen Zeiger auf den freien Diagnosepuffer besorgen. In diesen Puffer kann der Anwender dann seine Diagnose- oder Statusmeldungen (ab Byte 6) eintragen.

DPS2_GET_DIAG_BUF_PTR()		ersten Diagnosedatenpuffer holen
Übergabe	-----	
Rückgabe	UBYTE *	Zeiger auf Diagnosedatenpuffer; NIL wenn kein Diagnosepuffer mehr verfügbar.

4.2.13 Kontrolle der Baudrate

Mit dem Makro SPC3_SET_BAUD_CNTRL () läßt sich der Wurzelwert der Baudratenüberwachung einstellen. Nach der eingestellten Zeit (Wert x Wert x 10ms) beginnt der SPC3 selbständig mit der Baudratensuche, wenn in dieser Zeit kein plausibles Telegramm empfangen wurde. Wird vom Mastersystem der Watchdog verwendet, so wird zur Baudratenüberwachung der vom Master vorgegebene Wert für die Watchdogüberwachung verwendet. Wird der Slave ohne Watchdog betrieben, interpretiert der ASIC SPC3 den Eintrag des Wurzelwertes für die Baudratenüberwachung. Somit ist ein Zeitwert im Bereich von 20ms - 650s (Eintrag 2 - 255) möglich.

SPC3_SET_BAUD_CNTRL (x)		Baudratenüberwachung
Übergabe	UBYTE	Wurzelwert der Baudratenüberwachung
Rückgabe	------	

4.2.14 Start des SPC3

Mit SPC3_START schaltet sich der SPC3 online.

SPC3_START ()		SPC3 starten
Übergabe	------	
Rückgabe	------	

4.3 DPS2-Schnittstellenfunktionen

4.3.1 DPS2 - Benachrichtigungsfunktion (dps2_ind())

Die Interruptfunktion (hier mit dem Namen dps2_ind()) muß vom Anwender eingerichtet bzw. bereitgestellt werden. Der SPC3 löst einen Interrupt aus, sobald ein entsprechendes Ereignis eingetreten ist, welches in dem Interrupt-Bitfeld mittels dem Makro SPC3_SET_IND() freigegeben worden ist.

dps2_ind		Interruptfunktion
Übergabe	-------	
Rückgabe	-------	

Der Grund der Benachrichtigung kann vom Anwender durch unten aufgeführte Makros beim SPC3 abgefragt werden.

4.3.2 Benachrichtigungs-Grund auslesen

Mit dem Makro SPC3_GET_INDICATION() erhält der Anwender das Ereignis, das zur Benachrichtigung geführt hat (Interruptbit).

SPC3_GET_INDICATION()		Benachrichtigunsgrund auslesen
Übergabe	--------	
Rückgabe	UWORD	siehe das unter SPC3_SET_IND beschriebene Feld

Um die Leistungsfähigkeit vor allem des 803x, 805x zu steigern (Byte-orientiert) können Sie statt dessen auch jede Indication mit einem eigenen Makro (SPC3_GET_IND_...) erfragen. Dadurch kann eine laufzeit-optimale Schnittstelle geschaffen werden.

DPS2_GET_IND_GO_LEAVE_DATA_EX()	Die DP_SM ist in den Zustand 'DATA_EX' eingetreten oder hat ihn verlassen.
SPC3_GET_IND_MAC_RESET()	Der SPC3 ist, nachdem er den aktuellen Auftrag abgearbeitet hat, in den *Offline-Zustand* gekommen (durch Setzen des Bit 'Go_Offline')!!).
SPC3_GET_IND_BAUDRATE_DETECT()	Der SPC3 hat den 'Baud_Search-Zustand' verlassen und eine Baudrate gefunden.
DPS2_GET_IND_WD_DP_MODE_TIMEOUT	Im WD-Zustand 'DP_Control' ist der Watchdog-Timer abgelaufen.
SPC3_GET_IND_USER_TIMER_CLOCK	Die Zeitbasis des User_Timer_Clocks ist abgelaufen (1/10ms).
DPS2_GET_IND_NEW_GC_COMMAND()	Der SPC3 hat ein 'Global_Control-Telegramm' mit einem geänderten 'GC_Command-Byte' empfangen und dieses Byte in die RAM-Zelle 'R_GC_Command' abgelegt.
DPS2_GET_IND_NEW_SSA_DATA()	Der SPC3 hat ein 'Set_Slave_Address-Telegramm' empfangen und die Daten im SSA-Puffer bereitgestellt.
DPS2_GET_IND_NEW_CFG_DATA()	Der SPC3 hat ein 'Check_Cfg-Telegramm' empfangen und die Daten im Cfg-Puffer bereitgestellt.
DPS2_GET_IND_NEW_PRM_DATA()	Der SPC3 hat ein 'Set_Param-Telegramm' empfangen und die Daten im Prm-Puffer bereitgestellt.
DPS2_GET_IND_DIAG_BUFFER_CHANGED()	Auf Anforderung durch 'New_Diag_Cmd' hat der SPC3 die Diagnose-Puffer ausgetauscht und den alten wieder dem Anwender zur Verfügung gestellt.
DPS2_GET_IND_ DX_OUT()	Der SPC3 hat ein 'Write_Read_Data-Telegramm' empfangen und die neuen Output-Daten im -Puffer bereitgestellt. Bei 'Power_On' bzw. bei einem 'Leave_Master' löscht der SPC3 den Inhalt des Puffer und generiert auch diesen Interrupt..

Übergabe	--------	
Rückgabe	UBYTE	0/FALSE: kein Interrupt 1/TRUE: Diese Indication/Interrupt ist aufgetreten

4.3.3 Benachrichtigung quittieren

Das Makro SPC3_IND_CONFIRM() quittiert die durch dps2_ind() erhaltene Benachrichtigung.

SPC3_IND_CONFIRM(x)		Benachrichtig quittieren
Übergabe	UWORD	siehe das unter SPC3_GET_IND beschriebene Feld
Rückgabe	--------	

Auch hier ist eine Performance-Steigerung durch Definition je eines Makros für jede Benachrichtigung zu erreichen (siehe "Benachrichtigungs-Grund auslesen").

DPS2_CON_IND_GO_LEAVE_DATA_EX()		siehe oben
SPC3_CON_IND_MAC_RESET()		
SPC3_CON_IND_BAUDRATE_DETECT()		
DPS2_CON_IND_WD_DP_MODE_TIMEOUT		
SPC3_CON_IND_USER_TIMER_CLOCK		
DPS2_CON_IND_NEW_GC_COMMAND()		
DPS2_CON_IND_NEW_SSA_DATA()		
DPS2_CON_IND_DIAG_BUFFER_CHANGED()		
DPS2_CON_IND_ DX_OUT()		
Übergabe	--------	
Rückgabe	--------	

4.3.4 Benachrichtigung beenden

Das Makro SPC3_SET_EOI() beendet die Benachrichtigungssequenz / Interuptfunktion.

SPC3_SET_EOI()		Interrupt beenden
Übergabe	------	
Rückgabe	------	

4.3.5 Pollen der Benachrichtigung

Der Anwender kann Benachrichtigungen auch pollen statt sie sich mit dps2_ind() melden zu lassen. Dazu sind die Makros SPC3_POLL_IND_xx für ein einzelnes Auslesen oder SPC3_POLL_INDICATION() für ein globales Auslesen verfügbar. Gepollte Benachrichtigungen können ebenfalls mit dem Makro SPC3_IND_CONFIRM() quittiert werden.

SPC3_POLL_INDICATION()		Benachrichtigungsgrund
Übergabe	--------	
Rückgabe	UWORD	siehe das unter SPC3_SET_IND beschriebene Feld

DPS2_POLL_IND_GO_LEAVE_DATA_EX()	Die DP_SM ist in den Zustand 'DATA_EX' eingetreten oder hat ihn verlassen.
SPC3_POLL_IND_MAC_RESET()	Der SPC3 ist, nachdem er den aktuellen Auftrag abgearbeitet hat, in den *Offline-Zustand* gekommen (durch Setzen des Bit 'Go_Offline')
SPC3_POLL_IND_BAUDRATE_DETECT()	Der SPC3 hat den 'Baud_Search-Zustand' verlassen und eine Baudrate gefunden.
DPS2_POLL_IND_WD_DP_MODE_TIMEOUT()	Im WD-Zustand 'DP_Control' ist der Watchdog-Timer abgelaufen.
SPC3_POLL_IND_USER_TIMER_CLOCK()	Die Zeitbasis des User_Timer_Clocks ist abgelaufen (1/10ms).
DPS2_POLL_IND_NEW_GC_COMMAND()	Der SPC3 hat ein 'Global_Control-Telegramm' mit einem geänderten 'GC_Command-Byte' empfangen und dieses Byte in die RAM-Zelle 'R_GC_Command' abgelegt.
DPS2_POLL_IND_NEW_SSA_DATA()	Der SPC3 hat ein 'Set_Slave_Address-Telegramm' empfangen und die Daten im SSA-Puffer bereitgestellt.
DPS2_POLL_IND_NEW_CFG_DATA()	Der SPC3 hat ein 'Check_Cfg-Telegramm' empfangen und die Daten im Cfg-Puffer bereitgestellt.
DPS2_POLL_IND_NEW_PRM_DATA()	Der SPC3 hat ein 'Set_Param-Telegramm' empfangen und die Daten im Prm-Puffer bereitgestellt.
DPS2_POLL_IND_DIAG_BUFFER_CHANGED()	Auf Anforderung durch 'New_Diag_Cmd' hat der SPC3 die Diagnose-Puffer ausgetauscht und den alten wieder dem Anwender zur Verfügung gestellt.
DPS2_POLL_IND_ DX_OUT()	Der SPC3 hat ein 'Write_Read_Data-Telegramm' empfangen und die neuen Output-Daten im N-Puffer bereitgestellt. Bei 'Power_On' bzw. bei einem 'Leave_Master' löscht der SPC3 den N-Puffer und generiert auch diesen Interrupt.

Übergabe	--------	
Rückgabe	UBYTE	0/FALSE: kein Interrupt 1/TRUE: Diese Indication/Interrupt ist aufgetreten

4.3.6 Parametrierdaten prüfen

Die Funktion für die Prüfung der empfangenen Parametrierdaten muß der Anwender programmieren. Der SPC3 ruft die Interruptfunktion dps2_ind auf, in der durch NEW_PRM_DATA ermittelt werden kann, ob die Überprüfungsfunktion ausgeführt werden muß. Der benötigte Zeiger auf den entsprechenden Puffer sowie die Länge dieses Puffers können mit Makro-Aufrufen von DPS2 geholt werden.

Die Länge der erhaltenen Daten ermittelt das Makro DPS2_GET_PRM_LEN().

DPS2_GET_PRM_LEN ()		Parameterpufferlänge holen
Übergabe	--------	
Rückgabe	UBYTE	Länge der eingetroffenen Parametrierdaten

Einen Zeiger auf den aktuellen Parameterpuffer liefert DPS2_GET_PRM_BUF_PTR() .

DPS2_GET_PRM_ BUF_PTR()		Zeiger auf Parameterpuffer holen
Übergabe	--------	
Rückgabe	UBYTE *	Adresse des Parameterpuffers

Innerhalb dieser Überprüfungsfunktion hat der Anwender die Aufgabe, die erhaltenen User_Prm_Data auf Gültigkeit zu prüfen. Der Anwender quittiert die geprüften Parameter durch Aufruf des Makros DPS2_SET_PRM_DATA_OK() positiv, durch Aufruf von DPS2_SET_PRM_DATA_NOT_OK() negativ. Mit der Quittierung durch diese Makros wird zugleich die Interruptanforderung weggenommen, d. h. dieser Interrupt darf **nicht** mehr durch DPS2_IND_CONFIRM() quittiert werden. Der Rückgabewert der Makros muß wie unten beschrieben ausgewertet werden.

DPS2_SET_PRM_DATA_OK()		Die empfangene Parametrierung ist okay.
DPS2_SET_PRM_DATA_NOT_OK()		Dieses Makro teilt DPS2 mit, daß die Parametrierung nicht in Ordnung ist. Die übergebenen Parameter können im Gerät nicht verwertet werden.
Übergabe	--------	
Rückgabe	DPS2_PRM_FINISHED	Es liegt kein weiteres Parametriertelegramm vor => Ende der Sequenz
	DPS2_PRM_CONFLICT	Es liegt ein weiteres Parametriertelegramm vor! => Es muß eine nochmalige Überprüfung der Parametrierung erfolgen.
	DPS2_PRM_NOT_ALLOWED	Zugriff im derzeitigen Buszustand nicht erlaubt. Es könnte z. B. während der Überprüfung der Watchdog abgelaufen sein. Die Überprüfung der Parametrierdaten (und evtl. nachgeschaltene Funktionen in der Anwendung) sind zu verwerfen.

Achtung:
Bei Erhalt von Konfigurierung und Parametrierung **muß zuerst** eine Überprüfung der **Parametrierdaten** und deren Bestätigung erfolgen. Danach ist die Konfigurierung zu überprüfen. Die Reihenfolge ist zwingend vorgeschrieben.

4.3.7 Konfigurationsdaten prüfen

Die Funktion für die Prüfung der empfangenen Konfigurationsdaten muß der Anwender programmieren. DPS2 ruft die Funktion dps2_ind auf, in der der Anwender durch NEW_CFG_DATA ermittelt, ob die Überprüfungsfunktion auszuführen ist. Makro-Aufrufe von DPS2 liefern den benötigten Zeiger sowie die Länge des Puffers.

Die Länge der erhaltenen Daten ermittelt das Makro DPS2_GET_CFG_LEN().

DPS2_GET_CFG_LEN ()		Konfigurationspufferlänge holen
Übergabe	--------	
Rückgabe	UBYTE	Länge der empfangenen Konfigurationsbytes

Einen Zeiger auf den aktuellen Konfigurationspuffer liefert DPS2_GET_CFG_BUF_PTR() .

DPS2_GET_CFG_ BUF_PTR()		Zeiger auf Konfigurationspuffer holen
Übergabe	--------	
Rückgabe	UBYTE *	Adresse des Konfigurationspuffers

Innerhalb der Überprüfungsfunktion hat der Anwender die Aufgabe, die erhaltenen Cfg_Data mit den Real_Cfg_Data, d. h. seiner möglichen Konfiguration, zu vergleichen. Der Anwender quittiert die geprüften Konfigurationsdaten durch Aufruf des Makros DPS2_SET_CFG_DATA_OK() oder DPS2_SET_CFG_DATA_UPDATE() positiv, durch Aufruf von DPS2_SET_CFG_DATA_NOT_OK() negativ. Mit der Quittierung durch diese Makros wird zugleich die Interruptanforderung weggenommen, d. h. dieser Interrupt darf **nicht** mehr durch DPS2_IND_CONFIRM() quittiert werden. Der Rückgabewert der Makros muß wie unten beschrieben ausgewertet werden.

DPS2_SET_CFG_DATA_OK()		Die übergebene Konfiguration ist okay.
DPS2_SET_CFG_DATA_UPDATE()		Falls der Anwender wünscht, daß die geprüfte Konfiguration mit der bereits in DPS2 ausgetauscht werden soll, dann kann dies durch das Makro DPS2_SET_CFG_DATA_UPDATE() erreicht werden.
DPS2_SET_CFG_DATA_NOT_OK()		Dieses Makro teilt DPS2 mit, daß die Konfiguration nicht in Ordnung ist.
Übergabe	--------	
Rückgabe	DPS2_CFG_FINISHED	Es liegt kein weiteres Konfiguriertelegramm vor => Ende der Sequenz.
	DPS2_CFG_CONFLICT	Es liegt ein weiteres Konfiguriertelegramm vor! => Es muß eine nochmalige Überprüfung der eingetroffenen Konfigurierung erfolgen.
	DPS2_CFG_NOT_ALLOWED	Zugriff im derzeitigen Buszustand nicht erlaubt(es könnte z. B. während der Überprüfung der Watchdog abgelaufen sein). Die Überprüfung der Konfigurierdaten (und evtl. nachfolgende Funktionen in der Anwendung) sind zu verwerfen.

4.3.8 Übergabe der Output-Daten

DX_OUT in dps2_ind() zeigt empfangene Ausgabedaten an. Das Makro DPS2_OUTPUT_UPDATE() wechselt die Ausgabepuffer.

Das Makro DPS2_OUTPUT_UPDATE_STATE() liefert den Pufferzeiger und außerdem noch den Zustand des Dout-Puffers.

Die Länge der Ausgabedaten werden nicht bei jedem Update übergeben. Sie stimmt mit der durch DPS2_SET_IO_DATA_LEN() übergebenen Länge überein. Falls dies nicht der Fall wäre, würde DPS2 in den WAIT-PRM-Zustand zurückfallen.

DPS2_OUTPUT_UPDATE_STATE ()		Pufferzeiger und Zustand des Ausgangspuffer holen
Übergabe	UBYTE *	Zeiger auf Variable, in die der Status des Ausgangsdatenpuffers geschrieben werden soll
Rückgabe	UBYTE *	Zeiger auf den Ausgangsdatenpuffer oder NIL, wenn kein Puffer

In den Status (Zeiger auf diese Variable wurde übergeben) werden folgende Zustände (Bits) kodiert:

NEW_DOUT_BUF	Neue Ausgangsdaten erhalten
DOUT_BUF_CLEARED	Ausgangsdaten wurden gelöscht

DPS2_OUTPUT_UPDATE ()		Pufferzeiger auf Ausgangspuffer holen
Übergabe	------	
Rückgabe	UBYTE *	Zeiger auf den Ausgangsdatenpuffer oder NIL, wenn kein Puffer

4.3.9 Übergabe der Input-Daten

Vor dem ersten Eintrag seiner Eingangsdaten muß sich die Anwendung, wie beschrieben, mit dem Makro DPS2_GET_DIN_BUF_PTR() einen Puffer für die Eingabedaten holen.

Mit dem Makro DPS2_INPUT_UPDATE() können dann immer wieder die aktuellen Eingangsdaten vom Anwender an DPS2 übergeben werden. Die Länge der Eingaben wird nicht bei jedem Update übergeben, sie muß mit der durch DPS2_SET_IO_DATA_LEN() übergebenen Länge übereinstimmen.

DPS2_INPUT_UPDATE ()		Pufferzeiger auf Eingangspuffer holen und Pufferwechsel durchführen
Übergabe	------	
Rückgabe	UBYTE *	Zeiger auf den Eingangsdatenpuffer

Eine Umkonfigurierung der Eingangs-/Ausgangsdatenlänge kann mit den im Kapitel Initialisierung beschriebenen Funktionen und Makros (dps2_calculate_inp_outp_len(), DPS2_SET_IO_DATA_LEN(),...) erfolgen.

4.3.10 Diagnosedaten übergeben

Mit diesem Dienst kann der Anwender Diagnosedaten an DPS2 übergeben. Vor dem ersten Eintragen von externen Diagnosedaten muß sich der Anwender mit dem Makro DPS2_GET_DIAG_BUF_PTR() einen Zeiger auf den freien Diagnosepuffer besorgen. In diesen Puffer kann der Anwender dann seine Diagnose- oder Statusmeldungen (ab Byte 6) schreiben.

DPS2_GET_DIAG_BUF_PTR()		Zeiger auf Diagnosedaten Puffer holen
Übergabe	------	
Rückgabe	UBYTE *	Zeiger auf den Diagnosepuffer NIL wenn kein Diagnosedaten-Puffer im State ´U´

Die Länge der Diagnosedaten legt der Anwender durch Aufruf des Makros DPS2_SET_DIAG_LEN() fest. Die Länge darf erst eingestellt werden, nachdem erfolgreich ein Puffer durch DPS2_GET_DIAG_BUF_PTR() erhalten wurde.
Die Länge muß **immer** für den gesamten Puffer inclusive der durch die Norm festgelegten Bytes (+6) übergeben werden. Für den Fall, daß keine Anwenderdiagnose übertragen werden sollen, ist die **Länge 6** zu übergeben.

DPS2_SET_DIAG_LEN()		Länge der Diagnosedaten setzen
Übergabe	UBYTE	Länge der Diagnosedaten
Rückgabe	UBYTE	tatsächlich eingestellte Diagnoselänge 0xff, wenn kein Puffer dem Anwender zugeordnet ist

Der übergebene Zeiger von DPS2 zeigt auf Byte 0 des übergebenen Diagnosepuffers. Der Anwender darf seine Diagnose erst ab **Byte 6** in diesen Puffer eintragen. DPS2 trägt die festen Diagnosebytes (Byte 0 - 5) ein.

Struktur des zu übergebenden Datenblocks bei erweiterter Diagnose:

Byte	Diagnosedaten	Bemerkung
0	Stationsstatus_1	Byte 0 bis 5 fester Diagnose-Header
1	Stationsstatus_2	
2	Stationsstatus_3	
3	Diag.Master_Add	
4	Ident_Number_High	
5	Ident_Number_Low	
6 bis max. 241	Ext_Diag_Data	Beginn Anwenderdiagnose im Format der DP-Norm

Mit dem Makro DPS2_SET_DIAG_STATE() übergibt der Anwender DPS2 den neuen Diagnosestatus. Dieser muß vor dem Update der Diagnosedaten übergeben werden.

DPS2_SET_DIAG_STATE()		Setzen der Diagnosebits	
Übergabe	Bit	Bezeichnung	Bedeutung
	0	EXT_DIAG	Ist dieses Bit 1, dann wird das Diagnosebit Diag.Ext_Diag gesetzt, im anderen Fall wird das Bit zurückgesetzt.
	1	STAT_DIAG	Ist dieses Bit 1, dann wird das Diagnosebit Diag.Stat_Diag gesetzt, im anderen Fall wird das Bit zurückgesetzt.
	2	EXT_DIAG_OVF	Ist dieses Bit 1, so wird das Bit Diag.Ext_Diag_Overflow gesetzt, im anderen Fall wird Diag.Ext_Diag_Overflow zurückgesetzt
Rückgabe	------		

Mit dem Makro DPS2_DIAG_UPDATE() übergibt der Anwender DPS2 die neuen externen Diagnosedaten. Als Rückgabewert erhält er einen Zeiger auf den neuen Diagnosedatenpuffer.

DPS2_DIAG_UPDATE()		Diagnosedaten übergeben und neuen Zeiger holen
Übergabe	------	
Rückgabe	UBYTE *	Zeiger auf den Diagnosepuffer NIL wenn kein Diagnosedaten-Puffer vorhanden

In dem Fall, daß keine Diagnosedaten mit dem Makro DPS2_DIAG_UPDATE() übergeben werden sollen oder die zuvor übergebenen Diagnosedaten gelöscht werden sollen, muß die Diagnoselänge mit dem Makro DPS2_SET_DIAG_LEN() auf 6 gesetzt werden. Auf eine Diagnoseanforderung vom PROFIBUS DP-Master antwortet der SPC3 mit den 6 Byte Stationsdiagnosedaten.

4.3.11 Diagnosedaten-Puffer kontrollieren

Nach Übergabe der Diagnosedaten steht nicht automatisch der andere Wechselpuffer zur Verfügung. Der Anwender hat zwei Möglichkeiten, sich zu informieren, wann der Diagnosepuffer gesendet wurde:

- DPS2 meldet sich über die Benachrichtigungsfunktion dps2_ind() und zeigt das Ereignis mit DIAG_BUFFER_CHANGED an. Hierfür muß in der Initialisierung diese Benachrichtigungsfunktion freigegeben werden.
- Der Anwender pollt mit dem Makro DPS2_GET_DIAG_FLAG() den Zustand des Diagnosepuffers. Es zeigt an, ob der Puffer bereits gesendet wurde. Ist allerdings "statische Diagnose" eingestellt, wird immer der Zustand "Puffer nicht gesendet" zurückgemeldet.

DPS2_GET_DIAG_FLAG()		Zustand des Diagnosepuffer holen
Übergabe	------	
Rückgabe	UBYTE	TRUE: Diagnosepuffer noch nicht gesendet (oder statische Diagnose) FALSE: Der Diagnosepuffer ist bereits gesendet

4.3.12 Ändern der Slave-Adresse

Ein Anforderung zur Änderung der Slave-Adresse zeigt NEW_SSA_DATA an. Mit dem Makro DPS2_GET_SSA_BUF_PTR() kann ein Zeiger auf den Puffer mit der neuen Slave-Adresse und mit DPS2_GET_SSA_LEN() die Länge des erhaltenen SSA-Puffers ermittelt werden.

DPS2_GET_SSA_LEN()		Länge des Set_Slave_Address Puffers
Übergabe	------	
Rückgabe	UBYTE	Länge des SSA-Puffers

DPS2_GET_SSA_BUF_PTR()		Zeiger des Set_Slave_Address Puffers holen
Übergabe	------	
Rückgabe	UBYTE *	Adresse des SSA-Puffers

Der Anwender muß die Übernahme der Daten durch den Aufruf des Makros DPS2_SET_SSA_BUF_FREE() quittieren.

DPS2_SET_SSA_BUF_FREE()		Quittierung des Set_Slave_Address Dienstes
Übergabe	------	
Rückgabe	------	

4.3.13 Steuerkommandos melden

Diese Meldung zeigt das Eintreffen eines Global_Control-Telegramms an. Die Meldung erfolgt nur, wenn Gruppenzugehörigkeit und eine Änderung des Control_Commands gegenüber dem vorhergehenden Kommando erkannt wurde. Das Makro DPS2_GET_GC_COMMAND() liefert das Control_Command Byte. Damit hat der Anwender die Möglichkeit zusätzlich auf diese Kommandos zu reagieren. Die Abwicklung dieser Kommandos bezüglich der Pufferverwaltung wird von DPS2 intern durchgeführt, d. h. bei „Clear“ ein Löschen der Ausgangsdaten“.

DPS2_ GET_GC_COMMAND ()			Global Control Kommando holen
Übergabe	----		
Rückgabe	Bit	Bezeichnung	Bedeutung
	0	Reserved	
	1	Clear_Data	Mit diesem Kommando werden die Output-Daten gelöscht und dem Anweder zur Verfügung gestellt.
	2	Unfreeze	Mit "Unfreeze" wird das Einfrieren der Input-Daten aufgehoben.
	3	Freeze	Die Input-Daten werden "eingefroren". Neue Input-Daten werden erst wieder von der Anwendung geholt, wenn der Master das nächste 'Freeze'-Kommando sendet.
	4	Unsync	Das Kommando "Unsync" hebt das "Sync"-Kommando auf.
	5	Sync	Die letzten empfangenen Output-Daten werden der Anwendung zur Verfügung gestellt. Die nachfolgend übertragenen Output-Daten werden solange nicht zur Anwendung weitergereicht, bis das nächste 'Sync'-Kommando gegeben wird.
	6,7	Reserved	Die Bezeichnung "Reserved" gibt an, daß diese Bits für zukünftige Funktionserweiterungen reserviert sind

4.3.14 Verlassen des Data-Exchange-States

Die Meldung GO_LEAVE_DATA_EX zeigt an, daß DPS2 einen Zustandswechsel der internen State-Machine durchgeführt hat.

Mit dem Makro DPS2_GET_DP_STATE() bekommt die Anwendung die Information, ob DPS2 in den Data-Exchange-State eingetreten ist, oder diesen verlassen hat. Die Ursache dafür kann z. B. ein fehlerhaftes Parametriertelegramm in der Datentransfer-Phase gewesen sein.

DPS2_GET_DP_STATE()		Zustand der PROFIBUS DP Statemachine holen
Übergabe	------	
Rückgabe	DPS2_DP_STATE_WAIT_PRM	Warte auf Parametrierung
	DPS2_DP_STATE_WAIT_CFG	Warte auf Konfigurierung
	DPS2_DP_STATE_DATA_EX	Datenaustausch
	DPS2_DP_STATE_ERROR	Fehler

4.3.15 SPC3 Reset

Mit dem Makro SPC3_GO_OFFLINE geht der SPC3 in den Offline-Zustand. Dieser kann nur mit der SPC3_INIT-Funktion verlassen werden. Damit ist die Möglichkeit gegeben, neue Konfigurationsdaten zu übergeben und neu zu starten.

SPC3_GO_OFFLINE		Gehe zu Offline Zustand
Übergabe	-------	
Rückgabe	-------	

Ob der Übergang nach Offline durchgeführt wurde, kann mit Hilfe des Makros SPC3_GET_OFF_PASS() ermittelt werden.

SPC3_GET_OFF_PASS()		Prüfen des Offline Zustand
Übergabe	-------	
Rückgabe	UBYTE/Bit	1 = Passiv-Idle 0 = Offline

4.3.16 Ansprechüberwachung abgelaufen

WD_DP_MODE_TIMEOUT zeigt den Ablauf der Ansprechüberwachung an. Die PROFIBUS DP Statemachine verzweigt nach WAIT-PRM. .

Der Zustand der Watchdog-State-Machine kann mit dem Makro SPC3_GET_WD_STATE() erfragt werden.

SPC3_GET_WD_STATE()		Zustand der Watchdog-State-Machine
Übergabe	------	
Rückgabe	SPC3_WD_STATE_BAUD_SEARCH	Baudratensuche
	SPC3_WD_STATE_BAUD_CONTROL	Überprüfung der Baudrate
	SPC3_WD_STATE_DP_MODE	DP_Mode, d. h. Buswatchdog aktiviert

4.3.17 Neuparametrierung anfordern

Das Makro DPS2_USER_LEAVE_MASTER() veranlaßt DPS2/SPC3, in den State "Wait_Prm" zu wechseln.

DPS2_USER_LEAVE_MASTER()		in den Zustand Wait_Prm gehen
Übergabe	------	
Rückgabe	------	

4.3.18 Baudrate auslesen

Das Makro SPC3_GET_BAUD() liefert in kodierter Form die erkannte Baudrate.

SPC3_GET_BAUD()		Baudrate lesen
Übergabe	--------	
Rückgabe	BD_12M	12 MBaud
	BD_6M	6 MBaud
	BD_3M	3 MBaud
	BD_1_5M	1,5 MBaud
	BD_500k	500 KBaud
	BD_187_5k	187,5 KBaud
	BD_93_75k	93,75 KBaud
	BD_19_2k	19,2 KBaud
	BD_9_6k	9,6 KBaud

4.3.19 Adressierfehler feststellen

Der SPC3 meldet bei einem Zugriff oberhalb der 1.5KB des internen RAM einen Adressierfehler durch Anzeige von MAC_RESET und ACCESS_VIOLATION. Um den Adressierfehler vom Übergang zwischen "Offline" und "Passiv" unterscheiden zu können, ist zum einen das Makro SPC3_GET_OFF_PASS() und zum anderen das Makro SPC3_GET_ACCESS_VIOLATION() implementiert.

SPC3_GET_ACCESS_VIOLATION()		Adressierfehler eingetreten
Übergabe	------	
Rückgabe	UBYTE	≠ 0: Adressierfehler aufgetreten = 0: kein Adressierfehler

Achtung:
Im C32 Mode löst ein fehlerhafter Zugriff des Prozessors keinen Interrupt aus. Ein falscher Zugriff des SPC3 internen Mikrosequenzers führt jedoch zu einer Meldung.

4.3.20 Freien Speicherplatz im SPC3 ermitteln

Das Makro SPC3_INIT() richtet bei der Initialisierung Puffer im internen RAM des SPC3 ein. Mit diesem Makro kann der Anwender sich einen Zeiger auf den Beginn des freien Speicherplatzes im SPC3 sowie die Anzahl der noch verfügbaren Byte geben lassen. Diese Funktion gibt einen NULL-Zeiger zurück, wenn der SPC3 nicht korrekt initialisiert wurde.

SPC3_GET_FREE_MEM()		freien Speicherplatz ermitteln
Übergabe	UBYTE *	Zeiger auf die Zelle, in die der verfügbare Speicherplatz eingetragen wird
Rückgabe	UBYTE *	Zeiger auf den freien Speicherbereich im SPC3, 0, wenn SPC3 nicht korrekt initialisiert wurde

5 Beispielprogramm

5.1 Überblick

Das Beispielprogramm zeigt die Anwendung der DPS2-Software mit folgenden Beispielen:

- Die empfangenen Ausgangsdaten werden in einen definierten Speicherbereich (io_byte_ptr) abgelegt.
- Als Eingangsdaten wird dieser Speicherbereich zurückgelesen bzw. gespiegelt.
- Das erste Byte dieser Eingangsdaten beeinflußt die Diagnosebits in der bereits beschriebenen Weise.
- Der Beispielslave hat eine Einschaltkonfiguration von 0x13 / 0x23 (d.h. 4 Byte E/A) und kann sich an eine Konfiguration von 0x11/0x21 (d.h. 2 Byte E/A) anpassen. Inwieweit eine Änderung der Konfiguration sinnvoll ist, müssen Sie anhand Ihrer Anwendung entscheiden!
- Wenn in den userspezifischen Parameterdaten 0xAA und 0xAA stehen, meldet das Beispielprogramm eine Fehlparametrierung. Die userspezifischen Parameterdaten werden in das Feld der Diagnosedaten kopiert.

Sie können an den beschriebenen Schnittstellen Ihre Anwendung einhängen. Die zu bearbeitenden Programmodule sind im Directory user zusammengefaßt. Insbesondere müssen Sie die Stationsadresse über Ihren Mechanismus (z. B. Drehschalter, Tasten,...) ermitteln und eintragen. Eine eigene geräte-/herstellerspezifische PNO-Identnummer ist bei der PNO (siehe Anschriftenverzeichnis) erhältlich. Die Einbindung von eigenen Interruptprogrammen kann anwendungsabhängig in den im Sourcecode vorliegenden Interruptroutinen vorgenommen werden.

Für die Erzeugung ablauffähiger EPROMs sind im Diskettenverzeichnis Batchfiles, Commandfiles etc. beispielhaft dargelegt.

Der aktuelle Stand ist auf der Lieferdiskette gespeichert. Bitte beachten Sie aktuelle Implementierungshinweise auf der Mailbox des Schnittstellencenters (0911-737972).

5.2 Hauptprogramm

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

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


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

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

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


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


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

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

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

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

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


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


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

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


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

#ifdef __C51__
    HW_WATCHDOG_TRIGGER = 1;        /* Retrigger the HW Watchdog of the IM183*/
    HW_WATCHDOG_TRIGGER = 0;
```

```
#endif

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

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

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


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


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

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

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


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

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

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

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

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

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

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


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

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

                }

            dps_chg_diag_srvc_byte_old = dps_chg_diag_srvc_byte_new;

            }
        }
```

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

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

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

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

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

}


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


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


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

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

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


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

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


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

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

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

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

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


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


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

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

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

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

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

}
```


5.3 Interruptprogramm

Das folgende Interruptprogramm zeigt den prinzipiellen Ablauf des DPS2 Interruptprogramms in einer Anwendung.

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

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

{
UBYTE       i;

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

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

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

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

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

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

        } while(prm_result == DPS2_PRM_CONFLICT);

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

    }

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

    cfg_result = DPS2_CFG_FINISHED;
    result = DPS_CFG_OK;

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

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

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

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


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

                case DPS_CFG_FAULT: cfg_result = DPS2_SET_CFG_DATA_NOT_OK();
                    break;

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

        } while(cfg_result == DPS2_CFG_CONFLICT);
    }

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

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

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

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

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

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


6 Microcontroller-Implementierung

6.1 Entwicklungsumgebungen

Keil C51-Compiler ab Version 4.01
Boston Tasking C165-Compiler

6.2 Disketteninhalt

Die hardwareabhängigen Teile sind direkt als Unterfunktionen im Beispielprogramm oder in den anderen Funktionen des Userdirectorys ersichtlich.

Pfad	Datei	Beschreibung
user	userspc3.c	Anwenderprogramm mit main()
	intspc3.c	SPC3-Interrupt (nicht bei MINISPC3)
	dps2spc3.c	DPS2 Hilfsfunktionen (nicht bei MINISPC3)
	spc3dps2.h	Headerdatei
lst		Verzeichnis für Listings
obj	*.obj	übersetze Module
	*.hex	Hexfile für EPROM
prj	us.bat	Compileraufruf für userspc3.c
	it.bat	Compileraufruf für intspc3.c (nicht bei MINISPC3)
	d2.bat	Compileraufruf für dps2spc3.c (nicht bei MINISPC3)
	link.bat	Linker/Locatoraufruf
	spc3.l51	Linkercommandfile
	spc3.log	Ergebnisfile des Linker-/Locatorlaufes
	hex.bat	Aufruf des Object-Hex-Converters

6.3 Generierung

Die einzelnen Files im User-Verzeichnis können Sie mit Hilfe der Batches übersetzen und linken.
Dabei ist besonders zu beachten, daß der SPC3 auf die Hardwareadresse 0x1000 gelegt wird. Wenn durch eine entsprechende Beschaltung der SPC3 an eine andere Adresse gelegt wird, ist natürlich die Adreßzuweisung anzupassen

In den jeweiligen Files können Sie Anpassungen an Ihre Hardware oder Ihre Anwendung vornehmen. Die Interruptaufrufschnittstelle und die Bedienung der zugehörigen Steuerbits liegt Ihnen im Sourcecode vor, so daß Sie eigene Prozeduren einhängen können.

7 IM182-Implementierung

7.1 Entwicklungsumgebung

Die Software wurde mit folgenden Compilern getestet:

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

Die Verwendung von anderen Compilern sollte problemlos möglich sein.

7.2 Disketteninhalt

Pfad	Datei	Beschreibung
IM182	userspc3.c	Anwenderprogramm mit main()
	dps2spc3.c	DPS2 Hilfsfunktionen
	spc3dps2.h	Headerdatei
	spc3.ide	Projektdatei für Borland Compiler
	spc3msvc.mak	Projektdatei für Microsoft Compiler
	spc3wc.mak	Makefile für Watcom Compiler (16 bit DOS-Programm)
	spc3wc3.mak	Makefile für Watcom Compiler (32 bit DOS4GW Programm)

7.3 Generierung

Für Borland und Microsoft Compiler können Sie die Projektdateien in der jeweiligen IDE laden und das Programm erzeugen.
Für den Watcom Compiler rufen Sie einfach WMAKE mit dem gewünschten Makefile auf.

!!! ACHTUNG !!!
Für die 32-bit DOS4GW Variante müssen Sie in der Datei SPC3DPS2.H das Makro SPC3_FLAT definieren (die Rauskommentierung entfernen).

8 Anhang

8.1 Anschriften

PROFIBUS Nutzer Organisation

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

Technische Ansprechpartner bei ComDeC in Deutschland

Siemens AG
I IA SE DE DP3
Hr. Putschky

Briefadresse:
Postfach 2355
90713 Fürth

Hausadresse
Würzburgerstr.121
90766 Fürth

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

E-Mail:
Gerd.Putschky@siemens.com

Technische Ansprechpartner im PROFIBUS Interface Center in den USA

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

Fax : (423) - 262 - 2103

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

E-Mail:
profibus.sea@siemens.com

8.2 Allgemeine Begriffsdefinitionen

ASPC2	Advanced Siemens-PROFIBUS-Controller, 2te Generation
SPC3	Siemens-PROFIBUS-Controller, 3te Generation
SPM2	Siemens-PROFIBUS-Multiplexer, 2te Generation
LSPM2	Lean-Siemens-PROFIBUS-Multiplexer, 2te Generation
DP	Dezentrale Peripherie
MS	Mikrosequenzer
SM	State Machine

9 Anhang A: Diagnoseverarbeitung in PROFIBUS DP

9.1 Einführung

PROFIBUS DP bietet eine komfortable und vielschichtige Möglichkeit Diagnosemeldungen aufgrund von Fehlerzuständen zu verarbeiten.

Sobald eine Diagnoseanforderung erforderlich ist, antwortet der Slave im aktuellen Datenaustausch mit einem hochprioren Antworttelegramm. Der Master fordert daraufhin im nächsten Buszyklus von diesem Slave eine Diagnose an anstatt einen normalen Datenaustausch durchzuführen.
Ebenfalls kann jeder (nicht nur der zugegeteilte!) Master von dem Slave jederzeit eine Diagnose anfragen.
Die Diagnoseinformation des DP-Slaves besteht aus einer Standarddiagnoseinformation (6 Bytes) und kann durch userspezifische Diagnoseinformationen ergänzt werden.

Bei den ASICs SPM2 und LSPM2 ist eine weitgehende Diagnose durch entsprechende Beschaltung möglich. Bei den intelligenten SPCx Lösungen kann durch Programmierung eine angepaßte und komfortable Diagnoseverarbeitung durchgeführt werden.

9.2 Diagnosebits und erweiterte Diagnose

Teile der Standarddiagnoseinformation werden fest durch die Statemachine in der Firmware bzw. im Microprogramm der ASICs festgelegt.
Fordern Sie nur einmal Diagnose an ("update_diag(..)"), wenn ein Fehler vorliegt bzw. sich ändert. Keinesfalls sollte die Diagnose im Datenaustauschzustand zyklisch angefordert werden, da sonst das System durch redundante Daten belastet wird.
Drei Informationsbits können durch die Applikation beeinflußt werden:

9.2.1 STAT_DIAG

Der Slave kann aufgrund eines Zustandes in der Applikation keine gültigen Daten zur Verfügung stellen. Der Master fordert daraufhin nur noch Diagnoseinformationen an, solange bis dieses Bit wieder zurückgenommen wird. Die Statemachine ist im Zustand Data_Exchange, so daß sofort nach Rücknahme der statischen Diagnose der Datenaustausch aufgenommen werden kann.
Beispiel: Ausfall der Versorgungsspannung der Ausgangstreiber.

9.2.2 EXT_DIAG

Ist dieses Bit gesetzt, **muß** im user-spezifischen Diagnosebereich ein Diagnoseeintrag vorliegen. Ist dieses Bit nicht gesetzt, kann im user-spezifischen Diagnosebereich eine Statusmeldung vorliegen.

User-spezifische Diagnose
Die user-spezifische Diagnose kann in drei verschiedenen Formaten abgelegt werden:

Gerätebezogene Diagnose:

Die Diagnoseinformation kann beliebig kodiert werden.

	Bit 7	Bit 6	Bit 5-0
Headerbyte	**0**	**0**	Blocklänge in Bytes incl. Header
Diagnosefeld	Kodierung der Diagnose gerätespezifisch,.		
.....	kann nach Belieben festgelegt werden.		

Kennungsbezogene Diagnose:

Für jedes bei der Konfigurierung vergebene Kennungsbyte (z. B. 0x10 für 1 Byte Eingang) wird ein Bit reserviert.
Bei einem modularen System mit jeweils einem Kennungsbyte pro Modul kann damit modulspezifisch Diagnose angezeigt werden. Jeweils ein Bit zeigt dann Diagnose pro Modul an.

	Bit 7	Bit 6	Bit 5-0	
Headerbyte	**0**	**1**	Blocklänge in Bytes incl. Header	
Bitstruktur	1			1

⇑ Kennungsbyte 7 hat Diagnose usw. ⇑ Kennungsbyte 0 hat Diagnose

Kanalbezogene Diagnose:

In diesem Block werden der Reihe nach die diagnostizierten Kanäle und die Diagnoseursache eingetragen. Je Eintrag sind 3 Byte nötig.

	Bit 7	Bit 6	Bit 5	Bit 4 - 0
Headerbyte	**1**	**0**	Kennungsnummer	
Kanalnummer	Kodierung Ein-/Ausgang		Kanalnummer	
Art der Diagnose	Kodierung Kanaltyp			Kodierung Fehlertyp

Die Kodierung des Fehlertypes ist z. T. herstellerspezifisch, sonstige Kodierungen sind in der Norm festgelegt.
Beispiel:

0	**0**	0	0	0	1	0	0	**Geräte**bezogene Diagnose
Gerätespezifisches								Bedeutung der Bits
Diagnosefeld der								wird herstellerspezifisch
Länge 3								festgelegt
0	**1**	0	0	0	1	0	1	**Kennungs**bezogene Diagnose
							1	Kennungsnummer 0 hat Diagnose
			1					Kennungsnummer 12 hat Diagnose
						1		Kennungsnummer 17 hat Diagnose
1	**0**	0	0	0	0	0	0	**Kanal**bezogene Diagnose, Kennungsnummer 0
0	0	0	0	0	0	1	0	Kanal 2
0	0	0	1	0	1	0	0	Überlast, Kanal bitweise organisiert
1	**0**	0	0	1	1	0	0	**Kanal**bezogene Diagnose Kennungsnummer 12
0	0	0	0	0	1	1	0	Kanal 6
1	0	1	0	0	1	1	1	Oberer Grenzwert überschritten, Kanal wortweise organisiert

Status

Falls das Bit **EXT_DIAG** auf 0 gesetzt ist, werden die Daten aus Systemsicht heraus als Statusinformationen angesehen, z. B. Rücknahme des diagnoseauslösenden Fehlerfalles

9.2.3 EXT_DIAG_OVERFLOW

Dieses Bit wird gesetzt wenn mehr Diagnosedaten vorliegen als in den zur Verfügung stehenden Diagnosedatenbereich passen. Beispielsweise könnten mehr Kanaldiagnosen vorliegen als der Sendepuffer oder der Empfangspuffer ermöglicht.

9.3 Diagnoseverarbeitung aus Systemsicht

Die Diagnoseinformationen des Slaves werden, soweit sie busspezifisch sind, allein von der Masteranschaltung (z. B. IM308B) verwaltet.
Alle Diagnosen von der Anwendung werden dem S5 Programm über entsprechende Datenbytes zur Verfügung gestellt. Ist hierbei das **Bit externe Diagnose** gesetzt, kann bereits in der Diagnoseübersicht eine Auswertung der zu diagnostizierenden Slaves erfolgen. Hierauf kann eine spezielle Fehlerroutine aufgerufen werden. Es können dabei die Standarddiagnoseinformationen und die user-spezifischen Informationen ausgewertet werden.
Nach Beheben des vorliegenden Diagnosefalls, kann dies **ohne Setzen des Bits externe Diagnose** als Statusmeldung vom Slave gemeldet werden. Hierbei kann aus S5 Anwendersicht eine Auswertung erfolgen, der zurückgesetzte Diagnosefall wird jedoch bereits in der Diagnoseübersicht angezeigt.

Ein komfortables Diagnosewerkzeug steht mit dem COM ET200 online zur Verfügung. Damit können zur Zeit kennungsbezogene Diagnoseinformationen in Klartext angezeigt werden. In späteren Stufen werden auch kanalbezogene Diagnosen unterstützt. Die userspezifische Diagnose wird nur angezeigt, wenn das Bit EXT_DIAG gesetzt ist.

10 Anhang B: Nützliche Informationen

10.1 Datenformat in der Siemens SPS SIMATIC

Der SPC3 sendet immer alle Daten vom Pufferanfang bis Pufferende. 16 Bit Werte werden im Motorola Format übertragen.
Beispiel::

Buffer pointer	high byte
Buffer pointer +1	low byte

10.2 Aktuelle Anwenderhinweise

… finden sie in der Troubleshooting Liste unter www.ad.siemens.de/csi/pb-doc

10.3 PROFIBUS Literatur

PROFIBUS DP/DP-V1, Grundlagen, Tipps und Tricks für Anwender,
Hüthig Verlag, ISBN 3-7785-2781-9

Der neue Schnelleinstieg für PROFIBUS DP
Von DP-V0 bis DP-V2,
PROFIBUS Nutzerorganisation e. V., Haid und Neu Str. 7, 76131 Karlsruhe
Tel: 0721 9658 590
Best Nr. 4.071