



Produktübersicht

1 Einführung

Das Programmiersystem PROMAR NT dient der Programmierung, Simulation und Inbetriebnahme von speicherprogrammierbaren Steuerungen und Regelungen. Dabei ist PROMAR NT als offenes System zu verstehen, das an verschiedenste Steuerungen angepaßt werden kann, wenn diese eine Programmierung im Maschinencode erlauben.

1.1 Programmiersprachen

Die Programmierung erfolgt nach DIN IEC 1131-3. Aus diesem Standard werden die Teilsprachen

- Strukturierter Text (ST structure text) und
- Funktionsbaustein-Sprache (FBD function block diagram)

unterstützt. Weiterhin wird eine umfangreiche Modulbibliothek angeboten.

1.2 Hard- und Software-Voraussetzungen

PROMAR NT läuft auf allen IBM-kompatiblen PC's mit einem Prozessor I80386 oder besser. Weitere Voraussetzungen sind MS-Windows 95/98, wenigstens 32 MByte RAM insgesamt und etwa 5 MByte freier Festplattenspeicher. Für die Kommunikation mit einer SPS wird eine freie COM-Schnittstelle benötigt. Für die Übersetzung der Zwischensprachen C oder C++ in den ausführbaren Code wird ein kommerzieller C-Compiler benutzt.

1.3 Dokumentationen

Die Dokumentation zu PROMAR NT umfaßt folgende Handbücher:

- ☞ PROMAR NT Benutzerhandbuch
- ☞ PROMAR NT Programmierhandbuch
- ☞ PROMAR NT Handbuch zur Standardbibliothek
- ☞ PROMAR NT Systemhandbuch zur jeweiligen Steuerung

2 Funktionsumfang

Alle PROMAR NT-Komponenten werden in einer integrierten Entwicklungsumgebung verwaltet. Ihre Bedienoberfläche wurde mit MFC 4.2 von Microsoft entwickelt. Daraus resultieren:

- Eine SAA-konforme Bedienoberfläche;
- Mehrere Fenster, die beliebig positioniert und in der Größe variiert werden können;
- Mausunterstützung für jedes Tastaturkommando und Dialogfenster;
- Editierfunktion zum Zurücknehmen einer versehentlichen Textänderung;
- Online-Hilfen zur Bedienung, zur Programmiersprache und zu den Bibliotheksfunktionen;
- Die Möglichkeit häufig wiederkehrende Programmkonstrukte sowie Funktionsaufrufe aus dem Hilfefenster zu kopieren;

Wesentliche PROMAR NT-Komponenten sind:

- Die Projektverwaltung zur Verwaltung aller Dateien, die zur Erstellung eines Anwenderprogrammes benötigt werden. Hierzu gehört auch die von Scripts gesteuerte Übersetzung eines Projektes.
- Der ST-Editor zur Eingabe und Korrektur der textuellen Programmiersprache "Strukturierter Text".
- Der FBD-Editor zur Eingabe und Korrektur der grafischen Programmiersprache "Funktionsbaustein-Sprache".
- Die Compiler für die Übersetzung der Programmiersprachen in die Zwischensprache C++ (oder C) und die weitere Übersetzung bis zum Maschinencode.
- Der Simulator zum Test eines Anwenderprogrammes auf dem Entwicklungs-PC.
- Die Inbetriebnahme zum Laden und Testen eines Anwenderprogrammes auf der jeweiligen Steuerung.
- Die Identifikation mit dem Entwurf für einen PID-Regler.

3 Arbeitsbeschreibung

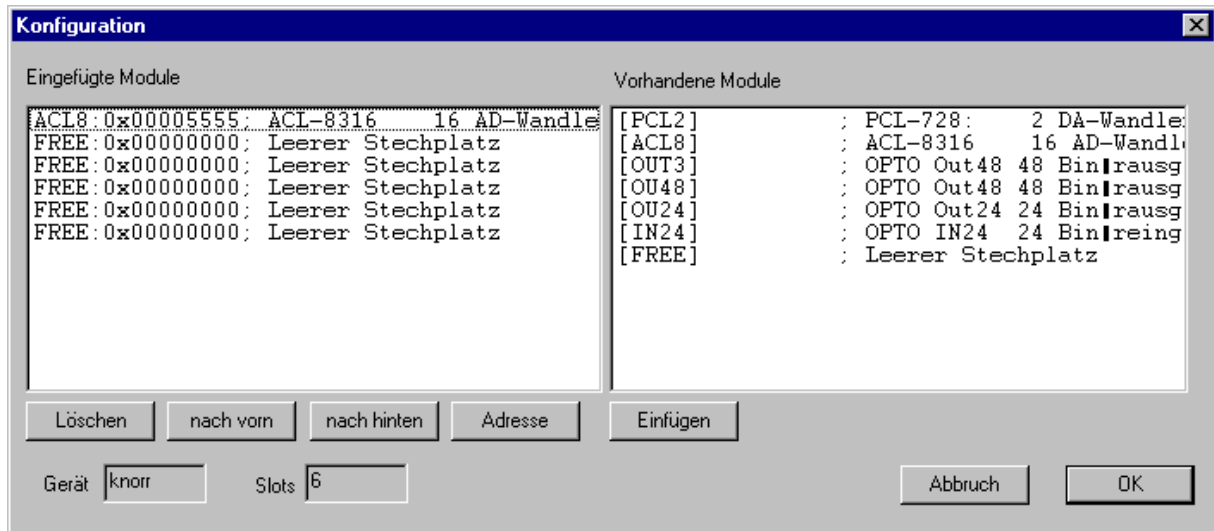
Promar NT dient der einfachen Bearbeitung komplexer Automatisierungsaufgaben. Es vereint in sich verschiedene Werkzeuge und unterstützt verschiedene Zielsysteme (vom Klein-Controller bis hin zum großen Industrie-PC). Alle Zielsysteme können innerhalb der Oberfläche programmiert - die erstellten Programme debuggt und simuliert werden. Weiterhin können verschiedene Programme mittels der Simulation verbunden werden, so daß auch geschlossene Strukturen überprüft werden können.

Die Auswahl des Zielsystems, der eingesetzten E/A-Hardware und weiterer grundlegender Einstellungen erfolgt im Projektdialog.

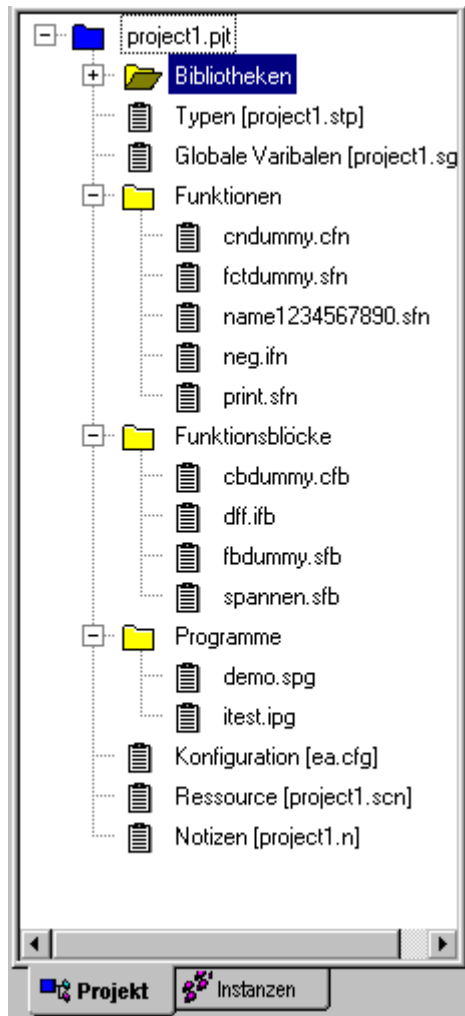
Unterstützte Klassen und Zielsysteme

	Zielsystem CPU	Betriebssystem	Kommunikations- server
IPC	80x86	µ RT (+ DOS)	RS 232 Ethernet
Klein-Controller	C16x	µ RT	RS 485
Simulation	80486 u. höher	µRT + Win32	

Nach der Auswahl einer Geräteklasse und eines speziellen Zielgerätetyps (z.B. IPC mit sechs freien Steckplätzen) kann eine genauere Beschreibung der Steckplatznutzung erfolgen. Die genaue Festlegung der verwendeten E/A-Module erfolgt mittels der Konfigurierung entsprechend der folgenden Dialogmaske. Aus einer Menge von unterstützten E/A-Modulen können die benötigten ausgewählt werden. Das Compilersystem wählt während der Erstellung die zugehörigen Treiber aus und integriert sie in das zu erstellende Anwenderprogramm.



Nach diesen grundlegenden Arbeiten beginnt die eigentliche Programmierung. In der vorliegenden Version kann zwischen den Sprachen Strukturierter Text, Anweisungsliste und purem C / C++ gewählt werden.



Mit dem Schließen der Projektdefinitionen wird durch das Entwicklungssystem ein fast vollständiges Programmgerüst erzeugt. Die erstellten Dateien werden übersichtlich in der Form eines Baums dargestellt.

Der Nutzer muß als mindeste Aktion einen leeren Programmrahmen erzeugen. Dazu wird über das Menü DATEI | NEU | PROGRAMME ein entsprechender Dialog geöffnet. Nachdem hier ein beliebiger Name (z.B. DEMO) eingetragen wurde und der Knopf OK gedrückt wurde, wird das gewünschte Programm erstellt. Wurde Strukturierter Text als Programmiersprache gewählt, so wird die Datei DEMO.SPG erzeugt.

Über das Menü ANSICHT | RESSOURCE muß nun nur noch eine Instanz des Programms angelegt werden. Dazu ist der Text **_programmname** in der unten abgebildeten Zeile durch den Namen des Programms (Demo) zu ersetzen.

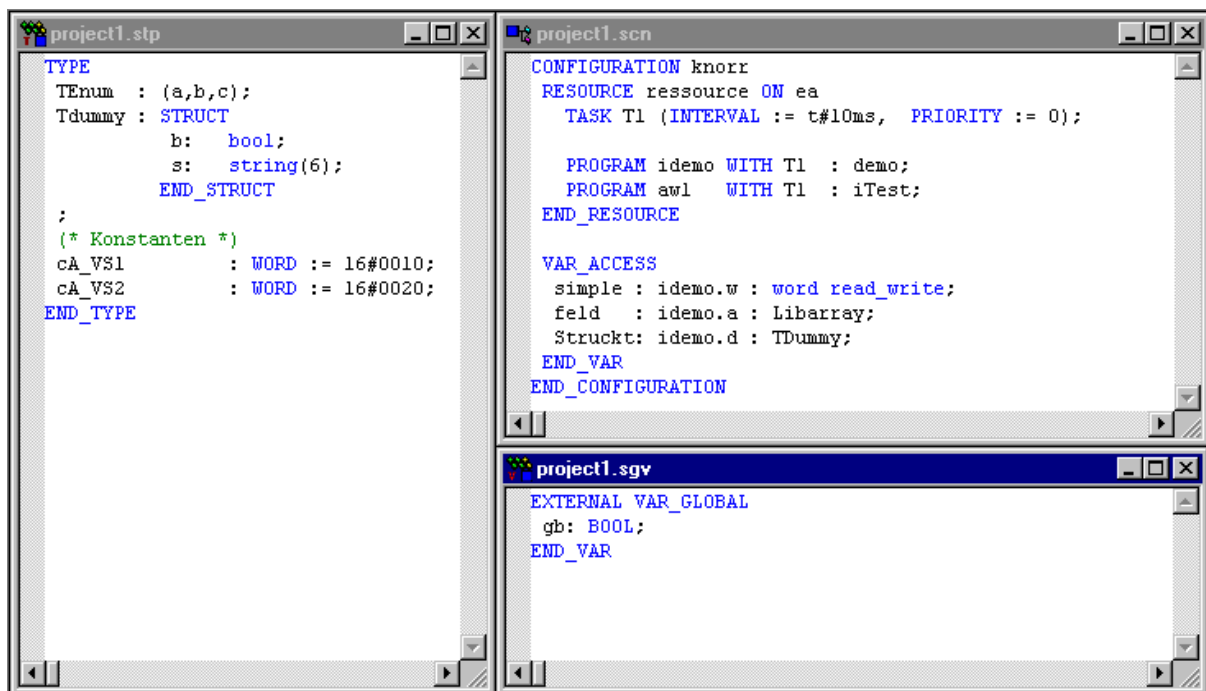
```
PROGRAM _instanzname WITH _taskname : _programmname;
```

Durch Drücken der Funktionstaste F7 wird das Programm übersetzt und eine abarbeitungsfähige Anwendung für das ausgewählte Zielsystem erstellt.

IEC - konforme Notation

In der folgenden Abbildung sind drei Fenster dargestellt, die nicht ursächlich den Programmalgorithmus betreffen. Dabei handelt es sich um:

- **Typfenster** – Hier können vom Anwender strukturierte Variablentypen definiert werden. Deren Nutzung im Programm trägt zur besseren Lesbarkeit bei.
- **Globale Variablen** – Deren Deklaration in einer separaten Datei trägt ebenfalls zur besseren Lesbarkeit bei. Auf diese Variablen kann durch alle Programme zugegriffen werden. Der Sichtbarkeitsbereich der Variablen wird nicht auf die Stelle ihrer Deklaration begrenzt.
- **Configuration** – Hier werden Instanzen der Programme angelegt. Außerdem können Festlegungen bezüglich deren Abarbeitung getroffen werden. Für den Zugriff auf das Anwenderprogramm von außen können Variablen in den ACCESS-Pfad eingetragen werden.



```
project1.stp
TYPE
TEnum : (a,b,c);
Tdummy : STRUCT
    b: bool;
    s: string(6);
END_STRUCT
;
(* Konstanten *)
cA_VS1 : WORD := 16#0010;
cA_VS2 : WORD := 16#0020;
END_TYPE

project1.scn
CONFIGURATION knorr
RESOURCE rresource ON ea
    TASK T1 (INTERVAL := t#10ms, PRIORITY := 0);

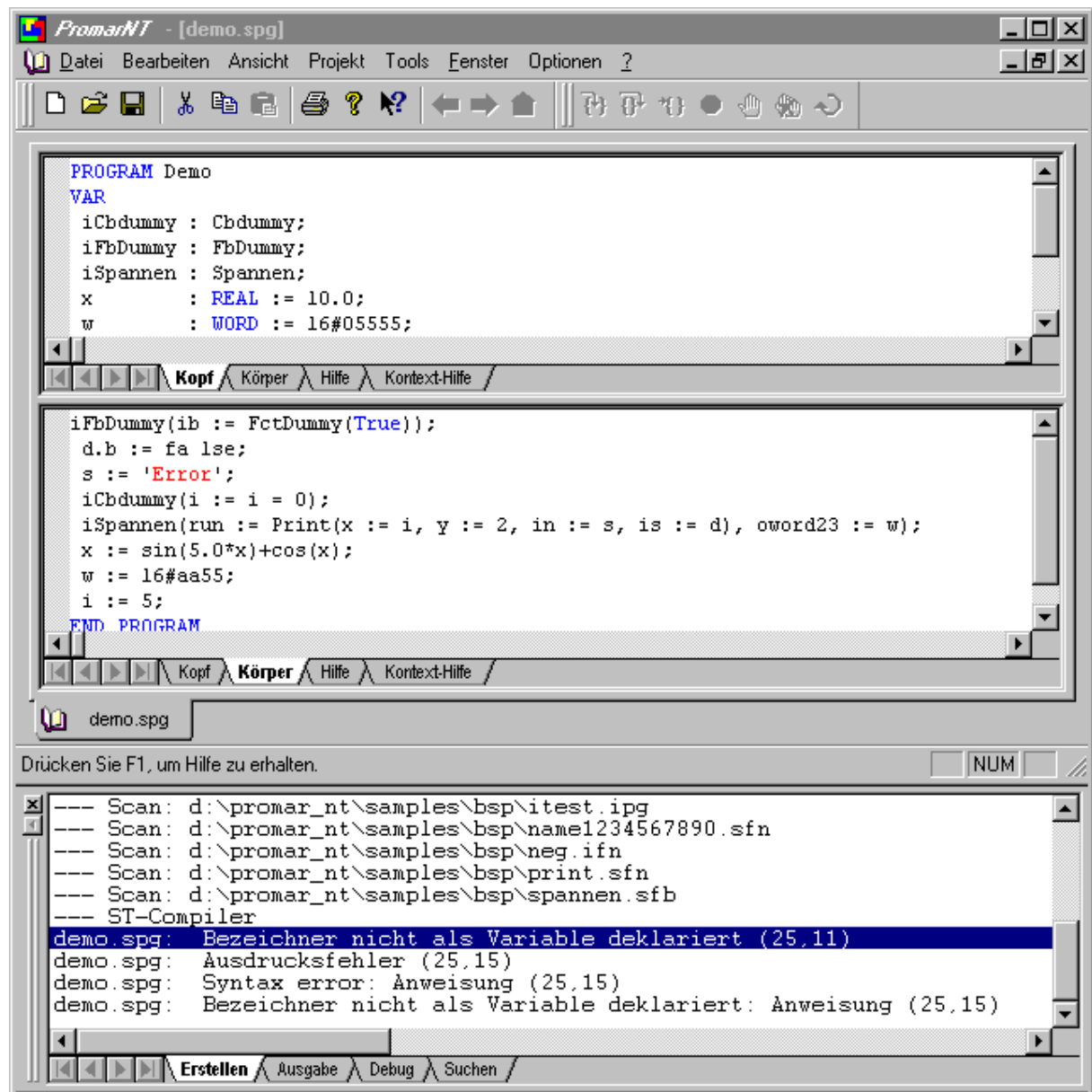
    PROGRAM idemo WITH T1 : demo;
    PROGRAM awl WITH T1 : iTest;
END_RESOURCE

VAR_ACCESS
simple : idemo.w : word read_write;
feld : idemo.a : Libarray;
Struckt: idemo.d : TDummy;
END_VAR
END_CONFIGURATION

project1.sgv
EXTERNAL VAR_GLOBAL
gb: BOOL;
END_VAR
```

Die Editor-Fenster

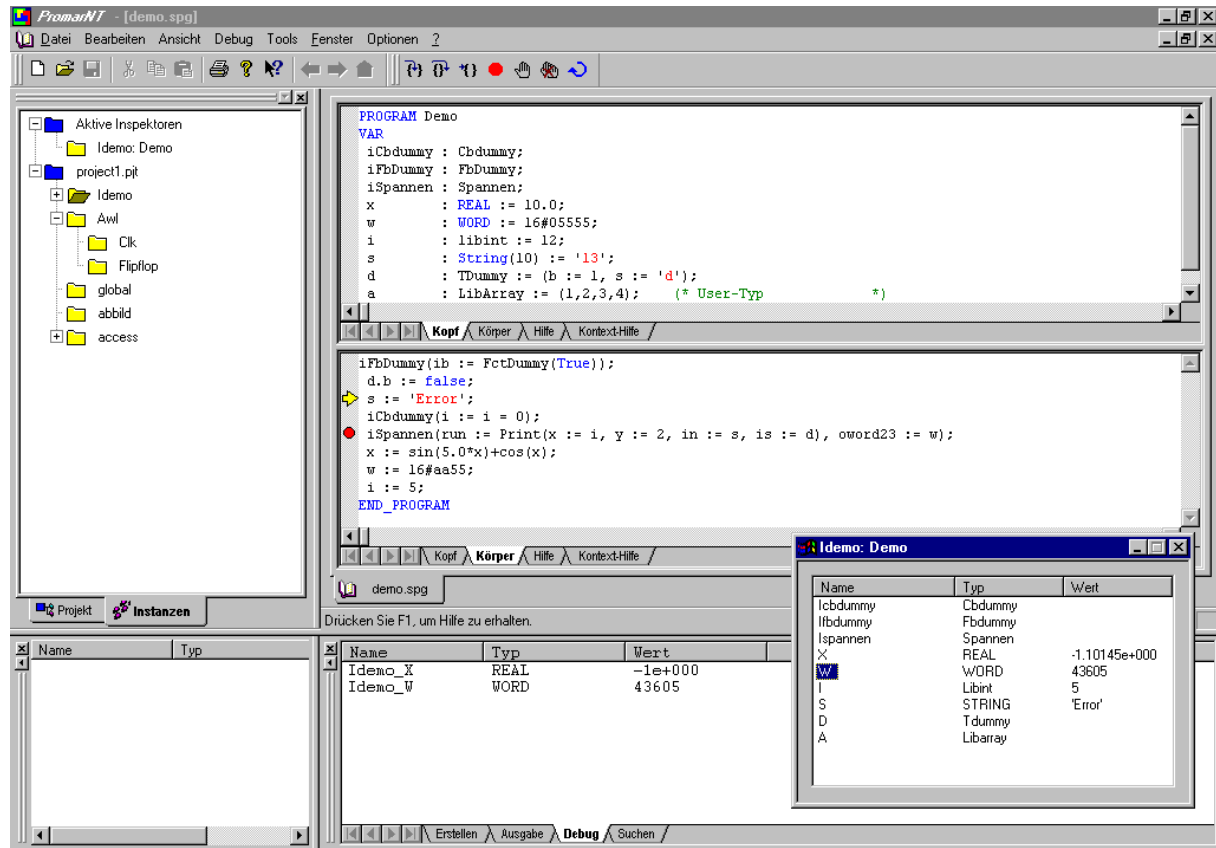
Das nachstehende Bild zeigt das Editorfenster. Ein Fenster für eine zu bearbeitende ST-Quelle besteht aus zwei Arbeitsflächen und jeweils vier Teilfenstern. In der oberen Arbeitsfläche wird nach dem Öffnen der Quelle der Kopf und in der unteren Arbeitsfläche der Körper dargestellt. Die Darstellung kann durch den Anwender durch Auswahl der Reiter geändert werden.



Nachdem der Quelltext eingegeben wurde, kann der Compiler aufgerufen werden. Dazu ist die Taste F7 zu drücken. Das gesamte Projekt wird daraufhin übersetzt. Fehler werden in der Ausgabebar dargestellt. Mittels Doppelclick auf die Fehlermeldung wird das Editorfenster geöffnet, in dem der Fehler durch den Compiler gefunden wurde.

Wurde das Projekt fehlerfrei übersetzt, kann in den Debug-Modus gewechselt werden (im Menü PROJEKT | DEBUG). Darauf hin werden die Debug-Funktionen aktiviert.

Aus dem Instanzenbaum können Elemente ausgewählt werden, die während der Programmabarbeitung angezeigt werden sollen.



Die Tasten F10 und F11 erlauben das schrittweise Ausführen von Anweisungen. Der gelbe Pfeil am linken Rand zeigt die aktuelle Position im Quelltext, d.h. die nächste auszuführende Anweisung.

Mit der Taste F9 können im Quelltext Haltepunkte gesetzt werden. Die Abarbeitung des Programms wird mit der Tastenkombination CTRL+F5 gestartet. Sobald ein Haltepunkt gefunden wird, wird die Abarbeitung unterbrochen. Gegebenenfalls kann die Abarbeitung auch mit der Tastenkombination SHIFT+F5 abgebrochen werden. Ein einzelner Zyklus wird mit der Taste F5 abgearbeitet.

Spezielle Variablen können in die Ausgabebar speziell in das Debug-Fenster eingefügt werden. Dazu ist im Inspektor die rechte Maustaste über der gewünschten Variablen zu drücken. Die Variablen werden nach jedem Programmschritt aktualisiert.

4 Strukturierter Text – Minimalbeispiel: (Projekt "demo1.pjt")

4.1 Konfiguration des Projektes:

Es wird festgelegt, daß das Projekt auf der Hardware-Konfiguration *demo_kon* (siehe unten: Belegung der E/A-Steckplätze) ablaufen soll. Ein Task mit Priorität 0 (höchste) und einem zyklischen Aufruf nach jeweils einer Sekunde wird vereinbart. Weiterhin wird eine Instanz IP1 des Programmes P1 (s.u.) gebildet und mit der Task T1 verknüpft. Das heißt, das Programm P1 wird zyklisch jede Sekunde aufgerufen und abgearbeitet. Dem Programm werden ferner die Ausgänge %QX2.1 bis %QX2.8 (Bit 1 bis 8 der Binärausgabe in Steckplatz 2) zur Benutzung unter den Variablennamen Ausgang1 bis Ausgang 8 zugewiesen. Als letztes erfolgt die Deklaration der Variablen, auf die von außen, z.B. von einem Visualisierungssystem, zugegriffen werden kann (VAR_ACCESS).

Es können natürlich weitere Instanzen des gleichen oder anderer Programme gebildet werden, die nicht nur zyklisch durch einen Timer, sondern auch durch asynchrone Ereignisse gestartet werden.

```
CONFIGURATION demo1

    RESOURCE ressource ON demo_kon

    TASK T1 (INTERVAL := t#1000ms, PRIORITY := 0);

    PROGRAM IP1 WITH T1 : P1(
        (* Binärausgaben Karte 1 *)
        Ausgang1 => %QX2.1, (* 1 *)
        Ausgang2 => %QX2.2, (* 2 *)
        Ausgang3 => %QX2.3, (* 3 *)
        Ausgang4 => %QX2.4, (* 4 *)
        Ausgang5 => %QX2.5, (* 5 *)
        Ausgang6 => %QX2.6, (* 6 *)
        Ausgang7 => %QX2.7, (* 7 *)
        Ausgang8 => %QX2.8 (* 8 *)
    );

    END_RESOURCE

    VAR_ACCESS
        Schritt_von_P1 : IP1.Schritt : INT READ_ONLY;
    END_VAR

END_CONFIGURATION
```

4.2 Belegung der E/A-Steckplätze (Datei *demo_kon.cfg*):

Diese Festlegung erfolgt in einem Auswahlmenü. Die Zielhardware knorr (spezieller IPC) und die Belegung der Steckplätze mit E/A-Karten, hier eine Binär-Ein und eine –Ausgabe, wird festgelegt.

```
[SPS]
knorr

[SLOT]
IN24:0x00000000; OPTO IN24 24 Binäreingänge bitweise organisiert
OU24:0x00000000; OPTO Out24 24 Binärausgänge bitweise organisiert
FREE:0x00000000; Leerer Steckplatz
FREE:0x00000000; Leerer Steckplatz
```

```
FREE:0x00000000; Leerer Steckplatz  
FREE:0x00000000; Leerer Steckplatz  
[]
```

4.3 Programm in ST – structured text (Datei p1.spg)

Im Programmkopf werden die verschiedenen Arten von Variablenamen und deren Typ, optional mit Initialwerten, vereinbart.

4.3.1 Programmkopf:

```
PROGRAM p1  
VAR Schritt : INT := 1;  
    a : INT := 2;  
    b : INT := 1;  
    c : INT;  
END_VAR  
  
VAR_OUTPUT  
    Ausgang1 : BOOL;  
    Ausgang2 : BOOL;  
    Ausgang3 : BOOL;  
    Ausgang4 : BOOL;  
    Ausgang5 : BOOL;  
    Ausgang6 : BOOL;  
    Ausgang7 : BOOL;  
    Ausgang8 : BOOL;  
END_VAR
```

Als nächstes folgt der Programmkörper. Hier erfolgt die Notation der eigentlichen Funktionalität. Am Anfang stehen 3 Zeilen Berechnungen. Durch die CASE-Schleife wird eine Taktkette einer Ablaufsteuerung realisiert, die nacheinander die Ausgänge 1 bis 8 anschaltet. Da hier keine Weiterschaltbedingungen (Taster oder Endlagenschalter an den Eingängen) definiert sind, wird das Programm (durch die Task-Steuerung aus der Konfiguration) jede Sekunde einen Ausgang weiterschalten.

4.3.2 Programmkörper:

```
c:= a*a + b*b;  
a:= a+1;  
b:= b*2;  
  
CASE Schritt OF  
1 : Ausgang1:=1;  
    Ausgang8:=0;  
    Schritt:=2;  
  
2 : Ausgang2:=1;  
    Ausgang1:=0;  
    Schritt:=3;  
  
3 : Ausgang3:=1;  
    Ausgang2:=0;  
    Schritt:=4;  
  
4 : Ausgang4:=1;  
    Ausgang3:=0;  
    Schritt:=5;  
  
5 : Ausgang5:=1;
```

```
        Ausgang4:=0;  
        Schritt:=6;  
  
6      :   Ausgang6:=1;  
        Ausgang5:=0;  
        Schritt:=7;  
  
7      :   Ausgang7:=1;  
        Ausgang6:=0;  
        Schritt:=8;  
  
8      :   Ausgang8:=1;  
        Ausgang7:=0;  
        Schritt:=1;  
  
END_CASE;  
  
END_PROGRAM
```

In praktischen Projekten werden natürlich weitere Programme vereinbart werden, die wiederum Funktionsbausteine und Funktionen (Unterprogramme und Prozeduren mit mehreren Eingängen und mehreren oder einem Ausgang) im Sinne der strukturierten Programmierung aufrufen können.

Neben Nutzer-Funktionen und –Funktionsbausteinen gibt es eine Bibliothek von Standard-Funktionen und –Funktionsbausteinen, in der z.B. die automatisierungstechnische Funktionalität realisiert wird. Beispiele sind Dynamikglieder (PT1, DT1, ...), Zweipunktregler, PID-Regler, Lose, usw.

5 Strukturierter Text – Erweitertes Beispiel für Modulares System mit C161 (Projekt "messe161.pjt")

Vorbemerkung

Das Beispiel MESSE161 benutzt alle vorhandenen Hardware-Komponenten ein und demonstriert einen kleinen Ausschnitt der Fähigkeiten von PROMAR NT.

Um das Beispiel erfolgreich zu verarbeiten, muß zuvor die zugehörige GNU-Bibliothek übersetzt werden. Das zu dieser Bibliothek gehörende Projekt befindet sich auf der CD im Verzeichnis GNU_LIB unterhalb der Beispiele (SAMPLES).

5.1 Konfiguration des Projektes:

*Es wird festgelegt, daß das Projekt auf der Hardware-Konfiguration **ea.cfg** (siehe unten: Belegung der E/A-Steckplätze) ablaufen soll. Ein Task mit Priorität 0 (höchste) und einem zyklischen Aufruf nach jeweils 500 ms wird vereinbart. Weiterhin wird eine Instanz **iDemo** des Programmes **Demo** (s.u.) gebildet und mit der Task T1 verknüpft. Das heißt, das Programm **Demo** wird zyklisch alle 500 ms aufgerufen und abgearbeitet. Dem Programm werden ferner die digitale Ausgänge %QX3.1 bis %QX3.8 und digitale Eingänge %IX1.1 bis %IX1.8 sowie analoge Eingänge %IW2.1 bis %IW2.8 zugewiesen.*

Als letztes erfolgt die Deklaration der Variablen, auf die von außen, z.B. von einem Visualisierungssystem, zugegriffen werden kann (VAR_ACCESS). Sie wird im Beispiel nicht benutzt.

Es können natürlich weitere Instanzen des gleichen oder anderer Programme gebildet werden, die nicht nur zyklisch durch einen Timer, sondern auch durch asynchrone Ereignisse gestartet werden.

```
CONFIGURATION messe161
  RESOURCE ressource ON ea
  TASK taskname (INTERVAL := t#500ms, PRIORITY := 0);
```

```
PROGRAM iDemo WITH taskname : Demo
(
  (* Analoge Eingänge *)
  Channels[1] := %IW2.1,
  Channels[2] := %IW2.2,
  Channels[3] := %IW2.3,
  Channels[4] := %IW2.4,
  Channels[5] := %IW2.5,
  Channels[6] := %IW2.6,
  Channels[7] := %IW2.7,
  Channels[8] := %IW2.8,

  (* Digitale Eingabe *)
  Taster1 := %IX1.1,
  Taster2 := %IX1.2,
  Taster3 := %IX1.3,
  Taster4 := %IX1.4,
  Taster5 := %IX1.5,
  Taster6 := %IX1.6,
  Taster7 := %IX1.7,
  Taster8 := %IX1.8,
```

```

(* Digitale Ausgabe *)
LED1 => %QX3.1,
LED2 => %QX3.2,
LED3 => %QX3.3,
LED4 => %QX3.4,
LED5 => %QX3.5,
LED6 => %QX3.6,
LED7 => %QX3.7,
LED8 => %QX3.8,

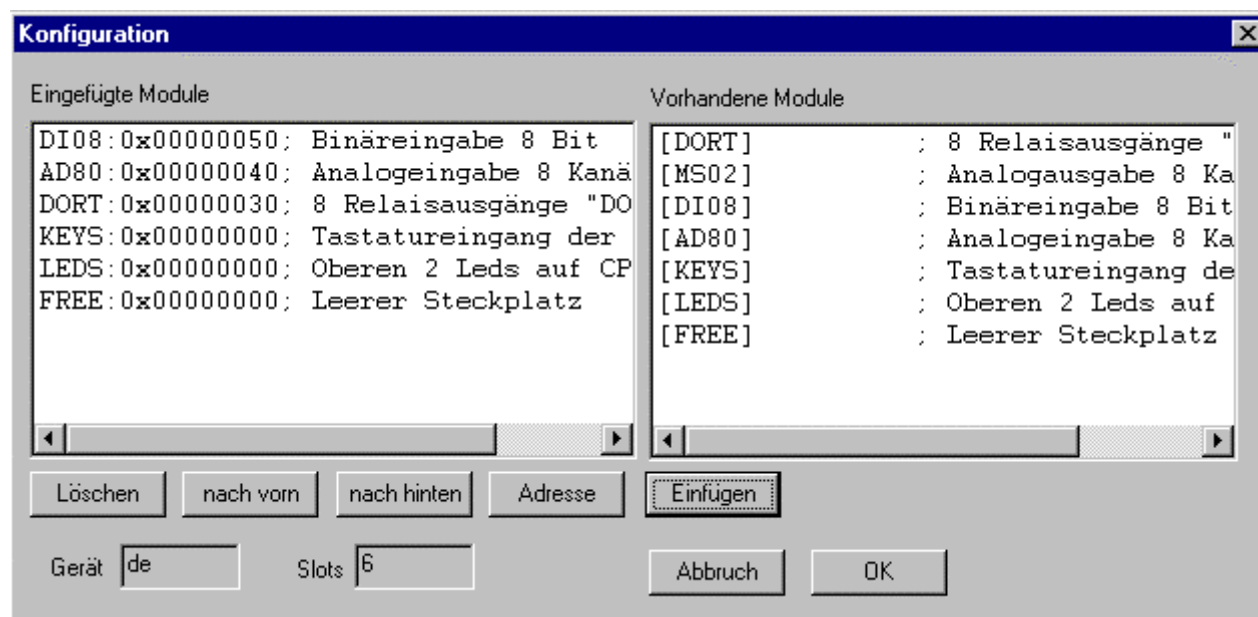
CPU_LED1 => %QX5.1,
CPU_LED2 => %QX5.2
);
END_RESOURCE

VAR_ACCESS
END_VAR
END_CONFIGURATION

```

5.2 Belegung der E/A-Steckplätze (Datei ea.cfg):

Das Projekt arbeitet mit allen zur Zeit zur Verfügung stehenden Baugruppen. In der nachstehenden Abbildung ist die Konfiguration des Systems dargestellt. Das 8-Bit-Binäreingabemodul steht an erster Stelle und besitzt die Adresse 0x50. In der zweiten Zeile folgt das 8-kanalige Analogeingabemodul mit Adresse 0x40. An der dritten Position sitzt das Relais-Ausgabe-Modul.



5.3 Programm in ST – structured text (Datei p1.spg)

Im Programmkopf werden die verschiedenen Arten von Variablennamen und deren Typ, optional mit Initialwerten, vereinbart. Der Programmkopf wurde nicht vollständig wiedergegeben. Der wesentliche Aufbau wird kurz umrissen.

5.3.1 Programmkopf:

```

PROGRAM Demo
  VAR_INPUT
    Taster1 : BOOL;

```

```
    Taster2 : BOOL;
...
END_VAR
VAR_OUTPUT
    LED1      : BOOL;
    LED2      : BOOL;
...
END_VAR
VAR
    isInit    : BOOL;      (* InitialFlag *)
    Mode      : TMode;
...
END_VAR
VAR_LOCAL
    N, I      : INT;
    V, B      : INT;
...
END_VAR
VAR
    memory : array [1..14000] of byte; (* Speichergrenze testen *)
END_VAR
```

5.3.2 Programmkörper:

Im Anschluß an den Programmkopf folgt der Programmkörper. Hier erfolgt die Notation der eigentlichen Funktionalität.

Das Hauptprogramm erlaubt die Auswahl zwischen verschiedenen Arbeitsmodi des Modularen Systems. Entsprechend des gewählten Betriebsmodus erfolgt z.B. im Modus Binärtest ein direktes Durchschalten der binären Eingänge auf die binären Ausgänge.

Weiterhin setzt das Programm drei nutzerdefinierte Funktionsbausteine ein (noise.sfb, speicher.cfb und textspeicher.sfb). Diese Modul liegen als Quelltext vor, die Dateien können über die Projektbar (Knoten Funktionsbausteine) geöffnet werden.

Der Baustein NOISE erzeugt Zufallszahlen. Der Baustein SPEICHER legt zusätzlichen statischen RAM über eine C-Anweisung an. Der Baustein TEXTSPEICHER erlaubt den indizierten Zugriff auf eine Texttabelle.