PS52

1-/2-Achs-CNC-Positioniersteuerung

PLC-Interface, CoDeSys 2.3
PS52.lib

Version 1.20
Stand 24.05.05

Diese Steuerung ist ein Produkt der

Multitron Elektronik GmbH u. Co. KG
Linsenhalde 11
D-71364 Winnenden
Tel.: 07195/9233-0
Fax.: 07195/63708
Internet: http://www.multitron.de
eMail: info@multitron.de

INHALT

1 Allgemein

Die **PS52** CNC-Positioniersteuerung kann optional mit einer integrierten SPS ausgestattet werden. Die SPS ermöglicht das Erstellen von SPS-Programmen in der IEC 61131 mit der Programmieroberfläche "**CoDeSys**". CoDeSys enthält alle die in der IEC 61131 genormten Editoren, wie

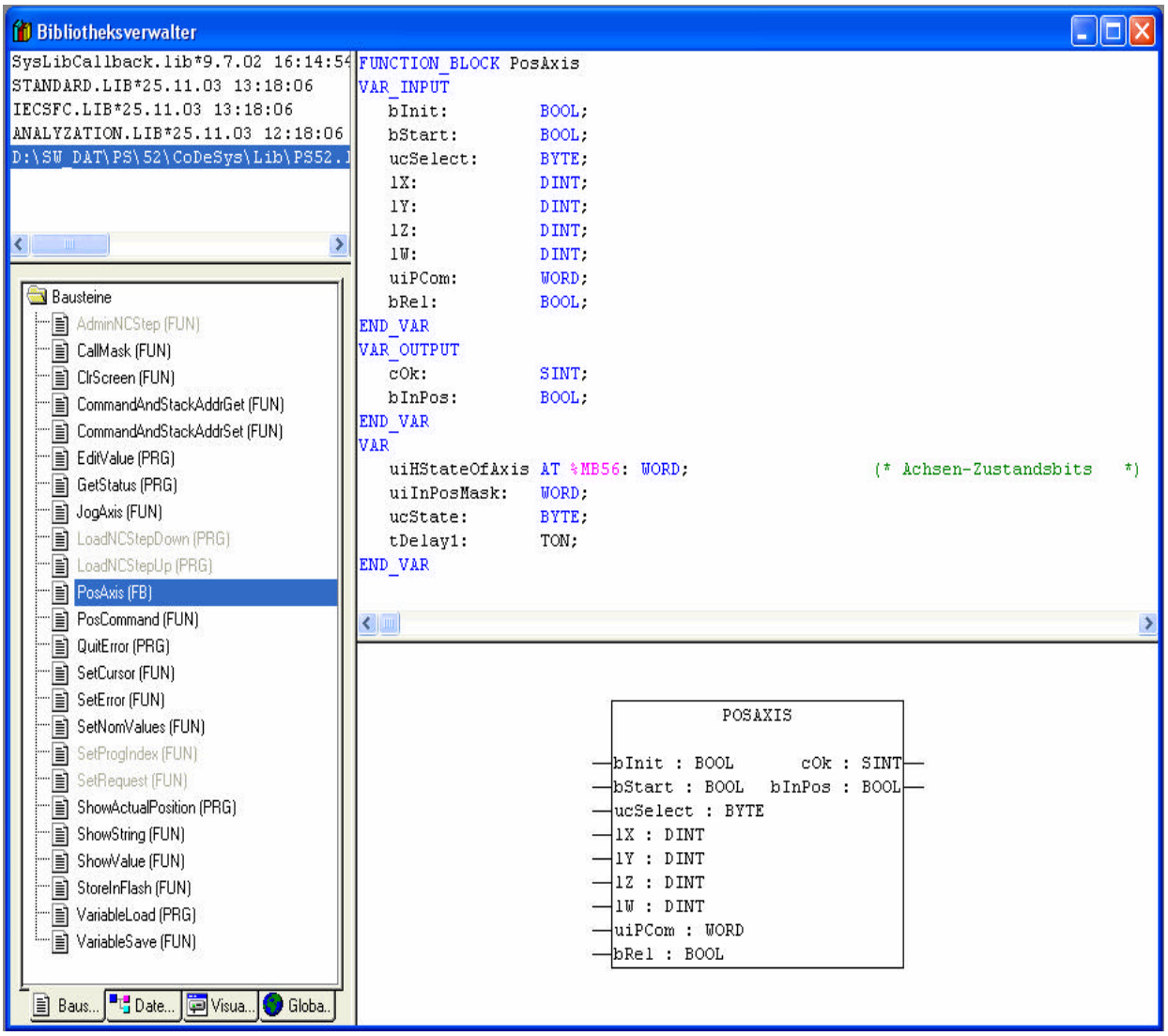
- Kontaktplan (KOP)
- Funktionsplan (FUP)
- Anweisungsliste (AWL)
- Strukturierter Text (ST)
- Ablaufsprache (AS)

Die PS52 stellt in dieser Konfiguration gleichzeitig eine CNC-Positioniersteuerung für bis zu 2 NC-Achsen und eine frei programmierbare SPS dar. Um nun die CNC- mit der SPS-Ebene zu verbinden, wurde eine Schnittstelle geschaffen, wodurch aus der SPS-Ebene bzw. dem SPS-Programm auf Funktionalitäten der CNC-Ebene zugegriffen werden kann. Die Steuerung der Bewegungsfunktionen kann somit komplett in der Hand des SPS-Programms liegen. Darüberhinaus besteht die Möglichkeit die Ausgaben auf dem Display frei zu gestalten und somit eigne Menü-Systeme zu erstellen. Die Tastatur-Codes der PS52 werden ebenso an die SPS übermittelt.

1.1 Einbinden der PS52.lib in die CoDeSys Programmieroberfläche

Um die Steuerungs- und peripheren Funktionen der PS52 von einem CoDeSys PLC-Programm ansprechen, bzw. nutzen zu können wurde die PS52.lib in Form einer CoDeSys-Library entwickelt. Die darin enthaltenen Funktionen stellen das Interface zur PS52 Steuerung her.
Um diese PS52.Library Funktionen zu verwenden, muss die Lib zunächst in der CoDeSys-Programmieroberfläche eingebunden werden. Dazu ist folgendes Vorgehen notwendig:

- ein CoDeSys Projekt muss geöffnet sein
- in CoDeSys, im Menüpunkt <FENSTER>, <BIBLIOTHEKSVERWALTUNG> aufrufen oder CoDeSys -> Ressourecen -> Bibliotheksverwalter
- u.a. Fenster öffnet sich
- im linken/oberen Fenster sind alle eingebundenen Librarys angegeben
- normalerweise befinden sich die CoDeSys-Librarys im Verzeichnis…
  `..\Programme\3S Software\CoDeSys 2.3\Library\..`
  dorthin kann auch die PS52.lib kopiert werden, muss aber nicht
- mit der Maus in dieses linke/obere Fenster navigieren
- rechte Maustaste klicken, es erscheint…
  `Weitere Bibliothek ... Einf.`
  nach Klicken dieser Funktion öffnet sich ein Auswahldialog, mit dem die neue Bibliothek in beliebigem Verzeichnis ausgewählt werden kann. Das Verzeichnis, indem sich PS52.lib befindet navigieren und auswählen.
- damit ist die PS52.lib eingebunden
- der Pfad für die PS52.lib wird in die Projektinformation aufgenommen. Wird das betr. Projekt erneut bei veränderter Verzeichnis-Struktur oder anderem Rechner geladen, kann die PS52.Lib u.U. nicht mehr gefunden werden. Das korrekte Verzeichnis muss dann wieder eingestellt werden.



2 PS52 Funktionen der CoDeSys PLC library

Das Interface zwischen CNC- und SPS-Ebene wurde durch eine Anzahl von Funktionen realisiert, die mit der CoDeSys Programmierumgebung erstellt wurden und in Form einer CoDeSys Library vorliegen. Um die Funktionen zu nutzen, muss diese Library (PS52.lib) zuerst in der Bibliotheksverwaltung der CoDeSys-Programmieroberfläche eingebunden werden. Man kann die zur Verfügung stehenden Funktionen in verschiedene Funkionsgruppen unterteilen.

- Funktionen zu Display-/Bildschirmbeeinflussung oder Gestaltung
- Tastaturabfrage
- Funktionen für die Bewegungssteuerung der NC-Achsen
- Funktionen zur Statusabfrage des PS52 internen CNC-Status
- Funktionen für die Verwaltung von remanenten Variablen
- Funktionen für die Verwaltung von NC-Programmen nach DIN66025
- Funktionen zur Fehlersignalisierung und -behandlung

Mit den vorliegenden Funktionen der PS52.Lib, eröffnet sich für den Anwender und CoDeSys-Programmierer die Möglichkeit einer vollkommen frei zu programmierenden Steuerung. Dabei liegen die Bildschirm- und Ablaufsteuerungen komplett in der Hand des SPS-Programms. Man kann jedoch auch Teile des bestehenden Menüsystems weiterverwenden, oder ein komplett eigenes Menüsystem aufbauen.
Die folgende Tabelle zeigt eine Übersicht aller bis dato zur Verfügung stehenden Funktionen in der PS52.lib mit einer Kurzbeschreibung auf.

3 Übersicht der Funktionen in der PS52.lib

Funktionsname	*Kurzbeschreibung*
AdminNCStep	Verwaltungsfunktion für Sätze innerhalb eines NC-Programms. - Satz einfügen - Satz löschen - ganzes CNC-Programm löschen
CallMask	Aufruf einer bestimmten Bildschirmmaske aus dem Laufzeitsystem oder aus dem PLC-Programm
ClrScreen	Bildschirm/Display löschen
EditValue	Eingabe/Editieren eines Zahlenwertes mit Angabe der Position im Display und Anzahl der Vor-/Nachkommastellen
GetMarkers	Abfrage der Zustände der Merker. 24 Merker stehen zur Verfügung über die eine Synchronisation mit einem NC-DIN-Programm erfolgen kann
GetStatus	Abfrage des Steuerungs-Gesamtstatus. Alle relevanten Steuerungszustände werden innerhalb einer Datenstruktur dargestellt
JogAxis	Jogbetrieb der NC-Achsen aus der PLC-Ebene
LoadNCStepDown	einen Satz eines NC-Programms aus der PLC-Ebene in den Programmspeicher der PS52 laden
LoadNCStepUp	einen Satz eines NC-Programms aus dem Programmspeicher der PS52 in die PLC-Ebene laden
PosAxis	die NC-Achsen auf einen bestimmten Positionssollwert mit einer bestimmten Interpolationsart fahren
PosCommand	ein bestimmtes Positionierkommando absetzen, z.B. NC-Achsen Stop, NC-Programm-Abarbeitung Start, Referenzfahrt NC-Achse, etc.
QuitError	eine anstehende Fehlermeldung quittieren
SetCursor	den Eingabe-Cursor am Display an einer bestimmten Displayposition ein- oder ausschalten
SetError	eine Fehlermeldung aus der PLC-Ebene erzeugen. Dadurch können eigene Fehlermeldungen aus dem PLC-Programm generiert werden. Die aus dem Betriebssystem der PS52 erzeugten Fehler sind davon unabhängig
SetMarkers	Setzen/Rücksetzen von Merkern. 24 Merker stehen zur Verfügung über die eine Synchronisation mit einem NC-DIN-Programm erfolgen kann
SetNomSpeeds	die Achsgeschwindigkeiten an die PS52 übertragen, bestimmen ob diese als Bahn- oder Achsgeschwindigkeiten interpretiert werden sollen
SetNomValues	die Achssollwerte an die PS52 übertragen, bestimmen ob diese als absolute oder relative Koordinaten interpretiert werden
SetProgIndex	die Programm- und Satznummer eines NC-Satzes innerhalb eines NC-Programms von der PLC-Ebene an die PS52 übertragen.
SetRequest	eine Anforderung einer best. Funktion von der PLC-Ebene an die PS52 absetzen
ShowActualPosition	Anzeige der aktuellen Istposition der NC-Achsen in einem bestimmbaren Format

ShowString	Ausgabe eines TextStrings auf dem Display an einer bestimmbaren Bildschirmposition
ShowValue	Ausgabe eines Zahlenwertes auf dem Display an einer bestimmbaren Displayposition in einem bestimmbaren Format
StoreInFlash	Speichern von remanenten Variablen und/oder Parameterwerten im FLASH (sp'ausfallsicheres Abspeichern)
VariableLoad	Laden einer remanenten Variable aus der PS52 in die PLC-Ebene
VariableSave	Speichern einer remanenten Variable aus der PLC-Ebene in die PS52

4 Die Funktionen der PS52.lib im Detail

4.1 AdminNCStep, Verwaltungsfunktion für Sätze eines NC-Programms

AdminNCStep

Beschreibung:

Diese Funktion ermöglicht die Beeinflussung von bestehenden oder neu anzulegenden CNC-Programmen nach DIN66025 in der PS52. Man kann die NC-Programstruktur verändern, d.h. NC-Sätze an bestimmten Stellen einfügen oder löschen, ganze NC-Programme löschen oder neue NC-Programme anlegen.
Für den Zugriff auf die NC-Programme sind dabei immer die Programmnummer und die Satznummer als Übergabeparameter erforderlich.

Input-Variablen:

`uiIPnr: WORD`
Die Variable *uiIPnr* ist vom Typ 16-Bit Wert und enthält die Programm-Nummer des NC-Programms, das mit dieser Funktion angesprochen werden soll. Dabei ist zu beachten, dass die PS52 intern 99 NC-Programme verwalten kann, die Programmnummer in der Variablen *uiIPnr* beginnt jedoch bei 0.
min/max Wert: 0..98
Um bspw. das NC-Programm 14 anzusprechen muss in *uiIPnr* 13 eingetragen werden.

`uiISnr: WORD`
Die Variable *uiISnr* ist vom Typ 16-Bit Wert und enthält die Satz-Nummer innerhalb des gewählten NC-Programms (*uiIPnr*), die mit dieser Funktion angesprochen werden soll. Dabei ist zu beachten, dass die PS52 intern 1000 NC-Sätze verwalten kann, die Satznummer in der Variablen *uiISnr* beginnt jedoch bei 0.
min/max Wert: 0..999
Um bspw. das NC-Programm 14 anzusprechen und innerhalb dieses Programms den Satz 37 muss in *uiIPnr* 13 und in *uiISnr* 36 eingetragen werden.

`uiAdminCode: WORD`
Die Variable *uiAdminCode* ist vom Typ 16-Bit Wert und enthält den Kommandocode. Dieser Code bestimmt die Aktion, die auf den durch *uiIPnr* und *uiISnr* adressierten NC-Satz ausgeführt werden soll.
min/max Wert: 1..3

1: NC-Satz an dieser Stelle einfügen, oder NC-Programm anlegen, falls noch nicht existent
2: Betr. NC-Satz löschen, oder NC-Programm ganz löschen, falls dies der einzige und letzte NC-Satz des Programms ist
3: Betr. NC-Programm komplett löschen

Output-Variablen:

`AdminNCStep: BOOL`
Die Funktion *`AdminNCStep()`* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Auswertung Tastatur, irgendeine Taste gedrückt ? *)
IF uiKeyCode<>0 THEN
   uiPnr:= 17; (* Programm 18 *)
   uiPnr:= 32; (* Satz 33 *)

   (* ENTER-Taste gedrückt ? *)
   IF uiKeyCode=KEY_ENTER THEN
      (* Satz an der Stelle uiPnr,uiSnr einfuegen *)
      AdminNCStep(uiPnr, uiSnr, 2);
   END_IF

   (* CLR-Taste gedrückt ? *)
   IF uiKeyCode=KEY_CLR THEN
      (* Satz an der Stelle uiPnr,uiSnr loeschen *)
      AdminNCStep(uiPnr, uiSnr, 1);
      IF GetStatus.uiNumOfSnr>0 THEN
         uiSnr:= GetStatus.uiNumOfSnr - 1;
      END_IF
   END_IF

   (* SHIFT-Taste gedrückt ? *)
   IF uiKeyCode=KEY_SHIFT THEN
      (* ganzes Programm uiPnr loeschen *)
      AdminNCStep(uiPnr, uiSnr, 3);
      uiSnr:= 0;
      IF GetStatus.uiNumOfSnr>0 THEN
         uiSnr:= GetStatus.uiNumOfSnr - 1;
      END_IF
   END_IF
END_IF
```

Bemerkung:

Dieses Kommando kann dazu genutzt werden, eine im PLC-Programm befindliche NC-Programmverwaltung aufzubauen. Um NC-Sätze mit Daten zu füllen, sind jedoch noch weitere Funktionen notwendig. Wichtig in diesem Zusammenhang sind auch *LoadNCStepDown(), LoadNCStepUp()* sowie *SetProgIndex()*.

Die Kommandos "Satz-Einfügen" oder "Satz-Löschen" funktionieren nur dann, wenn an der durch *uiPnr* und *uiSnr* adressierten Stelle auch ein CNC-Satz vorhanden ist. Man kann ein nicht vorhandenes CNC-Programm anlegen, indem man einen CNC-Satz mit der Satz-Nummer 1 (also uiSnr=0) einfügt. War das Programm noch nicht vorhanden, wird es durch diese Aktion angelegt.

4.2 CallMask, Bildschirmmaske aufrufen

CallMask

Beschreibung:

Diese Funktion ermöglicht den Aufruf einer bestimmten Bildschirmmaske. Jede Bildschirmmaske ist mit einer eindeutigen Nummer verbunden. Man kann somit an beliebiger Stelle die bereits existierenden Masken aus dem PS52 Betriebssystem oder neu zu generierende Masken aufrufen. Mit dem Aufruf einer Maske ist ein eindeutiger Steuerungszustand verknüpft. Die Steuerung verweilt also solange in diesem Zustand, bis eine andere Maske aufgrufen wird. Somit kann z.B. ein neuer Steuerungszustand definiert werden, in dem bspw. ein anwendungs-spezifischer Bewegungsablauf der NC-Achsen programmiert wird.

Input-Variablen:

uiMask: WORD

Die Variable *uiMask* ist vom Typ 16-Bit Wert und enthält die Nummer der aufzurufenden Bildschirmmaske. Innerhalb des Laufzeitsystems der PS52 existierenden Standard-Bildschirm-masken, über die die PS52 zu bedienen ist. Diese Masken sind normalerweise über ein definiertes Menüsystem verknüpft und werden darüber aufgerufen. Über das CoDeSys PLC-Interface kann nun auch aus dem PLC-Programm jede dieser existierenden Masken individuell aufgerufen werden. Darüberhinaus kann man eigene Masken generieren und beliebig in das Menüsystem integrieren.

Zuordnung der Nummern zu Bildschirmmasken in der PS52:

```
0000 : Haupt-Menu              (MAIN_MENU)
0001 : Referenz                (HOMING_MENU)
0002 : Manuell                 (MANUAL_MENU)
0003 : Automatik               (AUTOMAT_MENU)
0004 : Programmieren
0005 : Parameter
0006 : Diagnose-Menu
0007 : Service-Menu
0008 : Test digitale Eingänge
0009 : Test digitale Ausgänge
0010 : Test Inkrementalgeber-Eingang
0011 : Kontrasteinstellung Display
0012 : Test analoge Ausgänge
0013 : Test Ratemultiplier Frequenz-Ausgänge (Schrittmotor)
0014 : Tastatur-Test
0015 : Parameterwerte editieren
0016 : Parameterschalter einstellen
0017 : Passwort-Abfrage
0018 : Zustandsabfrage CoDeSys PLC-Programm
0019 : Sprache umschalten
0020 : Zustandsabfrage CNC-Programmspeicher
0021 : Abfrage der aktuellen Software-Version
0022 : Prozentuale Bewertung der Bewegungsgeschwindigkeiten
0023 : Einstellung der Optionen
0024 : Offset-Abgleich der Analog-Ausgänge
0025 – 0063 : reserviert für neue Masken bzw. für Masken aus dem PLC-
              Programm
```

uiType: WORD

Die Variable *uiType* ist vom Typ 16-Bit Wert und soll den Typ der aufzurufenden Bildschirmmaske beschreiben. Dieses Maskentyp hat bei der PS52 keine Relevanz; der darin übertragene Wert hat keine Auswirkung auf die Funktion. Bei anderen Displays stellt dieser Typ eine bestimmte Darstellungsart der Bildschirmmaske ein. Ob bspw. eine Bildschirmmaske mit Kopfzeile und/oder Funktionstastenbelegung oder nicht aufgebaut wird.
Bei der PS52 wird unabhängig vom Typ-Wort immer entweder eine bereits existierende oder im PLC-Programm generierte Maske gerufen.

Output-Variablen:

CallMask: BOOL

Die Funktion *CallMask()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Aufruf einer im Laufzeitsystem bestehenden Bildschirmmaske *)
IF uiKeyCode=KEY_ENTER THEN  (* ENTER-Taste gedrückt ? *)
   (* Aufruf der Bildschirmmask.AUTOMAT(=3), wenn noch nicht aktiv *)
   IF ucActMenu <> 3 THEN
      CallMask(3, 0);
   END_IF
END_IF

(* Aufruf einer im PLC-Programm generierten Bildschirmmaske *)
IF uiKeyCode=KEY_ENTER THEN  (* ENTER-Taste gedrückt ? *)
   (* Aufruf der Bildschirmmaske 26, wenn noch nicht aktiv *)
   IF ucActMenu <> 26 THEN
      CallMask(26, 0);
      ClrScreen(0); (* Bildschirm löschen *)
      ShowString(1, 1, 20, 0, '---<EIGENES MENU>---');
      ShowString(1, 2,  7, 0, 'Ist -X:');
      ShowString(1, 3,  7, 0, 'Soll-X:');
      ShowString(1, 4, 20, 0, 'noch Text in Zeile 4');
   END_IF
END_IF

(* das in der PS52 bestehende Menu-System soll weiterverwendet
   werden. Die Masken REFERENZ, MANUELL und AUTOMATIK jedoch auf
   auf eigene Masken umgeleitet werden *)

(*--- menu dispatcher *)
CASE ucActMenu OF
   1 : (* REFERENZ Maske wurde aufgerufen *)
      CallMask(25, 0); (* umleiten auf eigene Maske (25) *)

   2 : (* MANUELL Maske wurde aufgerufen *)
      CallMask(26, 0); (* umleiten auf eigene Maske (26) *)

   3 : (* AUTOMAT Maske wurde aufgerufen *)
      CallMask(27, 0); (* umleiten auf eigene Maske (27) *)

   25: (* eigene REFERENZ Maske, wird permanent gerufen *)
      MenuHoming;

   26: (* eigene MANUELL Maske, wird permanent gerufen *)
      MenuManual;

   27: (* eigene AUTOMAT Maske, wird permanent gerufen *)
      MenuAutomat;
END_CASE
```

Bemerkung:

In der globalen Variablendeclaration des mitgelieferten Beispielprogramms ist bei einer bestimmten Adresse im Übergabebereich des PLC-Interfaces ...

```
ucActMenu   AT %MB58:   BYTE;       (* aktuelle Menu-# *)
```

die Variable *ucActMenu* definiert. Diese Variable liefert immer die Nummer des aktuell aufgerufenen Menüs zurück. Mit dieser Information kann nun bei bestimmten Maskenaufrufen, auf eigene Masken verzweigt werden. Somit können Teile des bestehenden Menüsystems verwendet werden und nur an bestimmten Stellen auf neu generierte Masken im PLC-Programm verzweigt werden. Obiges Beispiel zeigt diese Art der Menüsystemverwaltung auf.

4.3 ClrScreen, Bildschirm/Display löschen

ClrScreen

Beschreibung:

Nach Aufruf dieser Funktionen wird der Bildschirm / das Display gelöscht. Die 4 Zeilen / 20 Zeichen des PS52-Displays werden mit Leereichen überschrieben.

Input-Variablen:

`uiAttr: WORD`

Die Variable *uiAttr* ist vom Typ 16-Bit Wert und enthält ein Attribut-Wort. Dieses Attribut-Wort hat bei der PS52 keine Relevanz; der darin übertragene Wert hat keine Auswirkung auf die Funktion. Bei anderen Displays stellt dieses Attribut eine bestimmte Löschfunktion des Bildschirms ein. Ob bspw. der Graphikbildschirm oder der Textbildschirm gelöscht werden soll, oder ob nur bestimmte Bereiche des Display gelöscht werden sollen.
Bei der PS52 wird unabhängig vom Attribut-Wort immer das ganze Display gelöscht

Output-Variablen:

`ClrScreen: BOOL`

Die Funktion *`ClrScreen()`* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Display löschen, einmalig *)
IF Programmzustand=1 THEN
   ClrScreen(uiAttr:=0);(* oder... *)
   ClrScreen(0);
   Programmzustand:= 2; (* nächster Programmzustand *)
END_IF
```

Bemerkung:

Beim Aufbau einer neuen Bildschirmmaske sollte zunächst das Display mit dieser Funktion gelöscht werden. Dadurch wird sichergestellt, dass keine Fragmente der bisherigen Anzeige stehenbleiben.
Siehe auch weitere Funktionen zur Bildschirmsteuerung…

- *CallMask(..)*
- *ShowString(..)*

4.4 EditValue, Zahlenwert in best. Format am Display editieren

EditValue

Beschreibung:

Mit der Funktion *EditValue()* kann an einer beliebigen Position (Displaykoordinaten) auf dem Display ein Zahlenwert in einem bestimmten Format editiert werden. Dabei sind folgende Dinge zu beachten:

- die Displaykoordinaten müssen sich innerhalb des zulässigen Bereichs für das Display befinden, also 4 Zeilen / 20 Zeichen
- die Stringlänge (Anzahl der Zeichen) des Zahlenwertes resultiert aus der angegebenen Formatierung. Dies ist bei der Wahl der Displaykoordinaten zu berücksichtigen. Überlange Strings werden abgeschnitten.
- die Funktion xx muss auch nach dem Start des Editors zyklisch aufgerufen werden.
- wird das Display gleichzeitig vom Laufzeitsystem mit anderen Textstrings an dieser Position beschrieben, konkurrieren die Systeme und editierte Zahlenwerte werden gegf. wieder überschrieben.

Input-Variablen:

bStartEdit: BOOL

Die Variable *bStartEdit* ist vom Typ Bool und startet die Editor-Funktion. Mit dem Start des Editors wird der übergebene, zu editierende Zahlenwert *lVal* zuerst einmalig angezeigt. Vor dem Setzen des Editor-Startflags müssen alle anderen Übergabe-Parameter auf entsprechende Werte initialisiert werden. *EditValue()* ist als PROGRAM ausgeführt; somit behalten die Übergabe-Parameter solange ihre Gültigkeit, bis sie wiederum verändert werden. Eine einmalige Initialisierung ist somit ausreichend.

uiX: WORD
uiY: WORD

Die Variablen *uiX* und *uiY* sind vom Typ 16-Bit Wert und enthalten die Displaykoordinaten. An dieser Stelle wird der Zahlenwert auf dem Display ausgegeben bzw. editiert. Das Display der PS52 hat 4 Zeilen mit je 20 Zeichen. Um einen Zahlenwert bspw. auf Zeile 3, Spalte 17 zu editieren, erfolgt der Aufruf so…

```
EditValue.uiX:= 17;
EditValue.uiY:= 3;
EditValue.uiPre:= 4;
EditValue.uiPost:= 0;
EditValue.bSign:= FALSE;
EditValue.bNotVis:= FALSE;
EditValue.bStart:= TRUE;
EditValue(lVal:=1234);
```

uiPre: WORD
uiPost: WORD

Die Variablen *uiPre* und *uiPost* sind vom Typ 16-Bit Wert und enthalten die Formatangabe des zu editierenden Zahlenwertes. *uiPre* stellt dabei die Anzahl der Vorkomma- *uiPost* die Anzahl der Nachkommastellen dar. Die max. Anzahl von Stellen ist 8. *uiPre* und *uiPost* addiert dürfen also keinen größeren Wert als 8 ergeben. Ist die editierte Zahl kleiner als sich aus der Formatangabe ergibt, werden führende Nullen mit Leerzeichen aufgefüllt. Die darzustellende Zahl ist immer ein Integer-Wert, die Komma-Stelle wird entsprechend der Formatangabe an der betreffenden Stelle gesetzt.

Wichtig: Zahlen vom Type float oder real können mit dieser Funktion nicht verarbeitet werden. Hier einige Beispiele für die formatierte Zahlenwertdarstellung:

```
EditValue.uiPre:= 4;
EditValue.uiPost:= 0;
EditValue(..., 1234);          Ausgabe: _1234

EditValue.uiPre:= 4;
EditValue.uiPost:= 0;
EditValue(..., -4);            Ausgabe: -___4
```

```
EditValue.uiPre:= 4;
EditValue.uiPost:= 3;
ShowValue(..., 123456);       Ausgabe: __123.456

EditValue.uiPre:= 2;
EditValue.uiPost:= 3;
ShowValue(..., -6789);        Ausgabe:   -_6.789

EditValue.uiPre:= 1;
EditValue.uiPost:= 3;
ShowValue(..., -123456);      Ausgabe:    -3.456
```

bSign: BOOL
Die Variable *bSign* ist vom Typ Bool und übergibt an die Editor-Funktion die Information, ob der zu editierende Zahlenwert auch negative Werte annehmen darf oder nicht.
`bSign:= FALSE, Eingabebereich Zahlenwert 0..max`
`bSign:= TRUE,  Eingabebereich Zahlenwert -max..max`
`mit max = 2`^{31} `- 1`

bNotVis: BOOL
Die Variable *bNotVis* ist vom Typ Bool und übergibt an die Editor-Funktion die Information, ob der zu editierende Zahlenwert sichtbar, also als Zahlenwert angezeigt wird oder versteckt bleibt, die editierten Zeichen werden als '*' angezeigt. Der verborgene Zahlenwert kann z.B. für Passwort-Abfragen o.ä. genutzt werden.
`bNotVis:= FALSE, Editierter Zahlenwert bleibt sichtbar`
`bNotVis:=  TRUE, Editierter Zahlenwert bleibt verborgen ("***")`

lVal: DINT
Die Variable *lVal* ist vom Typ 32-Bit Wert und enthält den zu editierenden Zahlenwert. Es handelt sich also um einen 32-Bit vorzeichenbehafteten Integer-Wert. Andere Typen können von *EditValue()* nicht verrabeitet werden.

Output-Variablen:

lEditValue: DINT
Die Funktion *`EditValue()`* liefert als Rückgabewert einen 32-Bit-Wert, signed DINT, der den editierten Zahlenwert enthält. Dieser Wert ist nur dann gültig, wenn *`bNewEdVal`* =TRUE ist.

bNewEdVal: BOOL
Die Funktion *`EditValue()`* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn die Editorfunktion durch eine Zahlengabe beendet wurde und sich in *`lEditValue`* ein gültiger Wert befindet.

Beispiel:

```
(*--- Zustandsmaschine *)
CASE ucState OF
   (* Bildschirmmaske aufbauen *)
   STATE_MASK:
      ClrScreen(0);
      SetCursor(0, 0, 0); (* Cursor ausschalten *)
      lAbs:= 12345;
      ucState:= STATE_SHOW_DATA;

   (* zu editierenden Zahlenwert anzeigen *)
   STATE_SHOW_DATA: (* show data *)
      ShowValue(8, 3, 4, 1, 0, lAbs);
      ucState:= STATE_PRE_EDIT;

   (* Editor-Parameter einstellen *)
   STATE_PRE_EDIT:
      EditValue.uiX:= 8;
      EditValue.uiY:= 3;
      EditValue.uiPre:= 4;
      EditValue.uiPost:= 1;
      EditValue.bSign:= FALSE;
      EditValue.bNotVis:= FALSE;
      EditValue.lVal:= lAbs;
      EditValue.bStartEdit:= TRUE;
      ucState:= STATE_EDIT;

   (* Editor zyklisch aufrufen, Zahlenwert eingeben *)
   STATE_EDIT:
      EditValue();
      (* Eingabe fertig, editierter Wert steht zur Verfügung *)
      IF EditValue.bNewEdVal THEN
         lAbs:= EditValue.lEditValue;
         ucState:= STATE_SHOW_DATA;
      END_IF
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Beeinflussung von Zahlenwerten bzw. Bildschirmdarstellung

- *ShowValue(..)*
- *ShowActualPosition(..)*

Der Eingabe-Cursor, dessen Erscheinungsbild auch durch die Funktion ***`SetCursor()`*** beinflussbar ist, wird durch die Funktion ***`EditValue()`*** selbständig ein- bzw. umgeschaltet. Mit dem Setzen des ***`bStartEdit`*** Flags und dem ersten Aufruf von ***`EditValue()`*** wird der Eingabe-Cursor automatisch eingeschaltet.

4.5 GetMarkers, Merkerzustände abfragen

GetMarkers

Beschreibung:

Mit der Funktion ***GetMarkers()*** besteht die Möglichkeit, die Zustände der Merker in der PS52 abzufragen. 24 Merker stehen in der PS52 zur Verfügung. Da Merker auch im NC-Programm beeinflussbar sind, kann damit eine Synchronisation eines PLC- und NC-Programm-Ablaufs erfolgen.
Beispielsweise kann eine bestimmte Aktion in einem NC-Programm über einen Merkerzustand an das PLC-Programm, umgekehrt kann aus dem PLC-Programm über einen Merker ein bestimmter PLC Zustand an das NC-Programm gemeldet werden. Beide Programme (PLC und NC), die normalerweise unabhängig voneinander ablaufen, kann man auf diese Art der Merkerverarbeitung in eine gegenseitige Abhängigkeit bringen.

Input-Variablen:

Output-Variablen:

abMarker: ARRAY [1..24] OF BOOL
Das Variablenfeld *abMarker* enthält die aktuellen Zustände aller verfügbaren 24 Merker und ist vom Typ BOOL. Zu Aktualisierung der Merker-Zustände kann die Funktion ***GetMarkers()*** zylisch im PLC-Programm gerufen werden. Jedes Element des Variablenfeldes enthält den bool'schen Zustand (TRUE oder FALSE) des entsprechenden Merkers.

bOk: BOOL
Die Funktion ***GetMarkers()*** liefert als Rückgabewert auch einen bool'schen Wert, der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte und die Merker-Zustände aus der PS52 geladen werden konnten.

```
bOk=TRUE  (* Merkerzustände konnten von PS52 geladen werden *)
```

Beispiel in STRUCTURED TEXT :

```
(*--- alle 24 Merker init./ruecksetzen *)
IF bInit THEN
   FOR ucN:=1 TO 24 DO
      SetMarkers(ucN, FALSE);
   END_FOR
   A03:= FALSE;
   bInit:= FALSE;
END_IF

(*--- alle 24 Merkerzustaende laden *)
GetMarkers;
IF GetMarkers.bOk THEN
   (* Visualisierung des Merkerzustandes-3 durch dig. Output-3 *)
   A03:= GetMarkers.abMarker[3];

   (* Merkerzustand-3 toggeln mit dig. Input-3 *)
   E03_T(CLK:=E03);
   IF E03_T.Q THEN
      SetMarkers(3, NOT GetMarkers.abMarker[3]);
   END_IF
END_IF
```

Bemerkungen:

siehe auch weitere Funktionen zur Merkerverarbeitung…

- *SetMarkers(..)*

4.6 GetStatus, Steuerungs-Gesamtstatus abfragen

GetStatus

Beschreibung:

Mit dieser Funktion wird die Abfrage des Gesamtstatus der PS52 Steuerung ermöglicht. Im Gesamtstatus sind alle wesentlichen Stati und Informationen zur Steuerung eines Bewegungsablaufes enthalten. Neben bitcodierten Zuständen findet man auch numerische Werte, wie z.B. Positionsist- und -sollwerte.
Zur Aktualisierung der Daten muss die Funktion zyklisch im SPS-Programm aufgerufen werden. Die Funktion ist in der IEC Terminologie als PROGRAM ausprogrammiert, enthält keine Eingangs-Parameter und liefert eine Struktur von Ausgangs-Parametern, die den Gesamtstatus enthält.

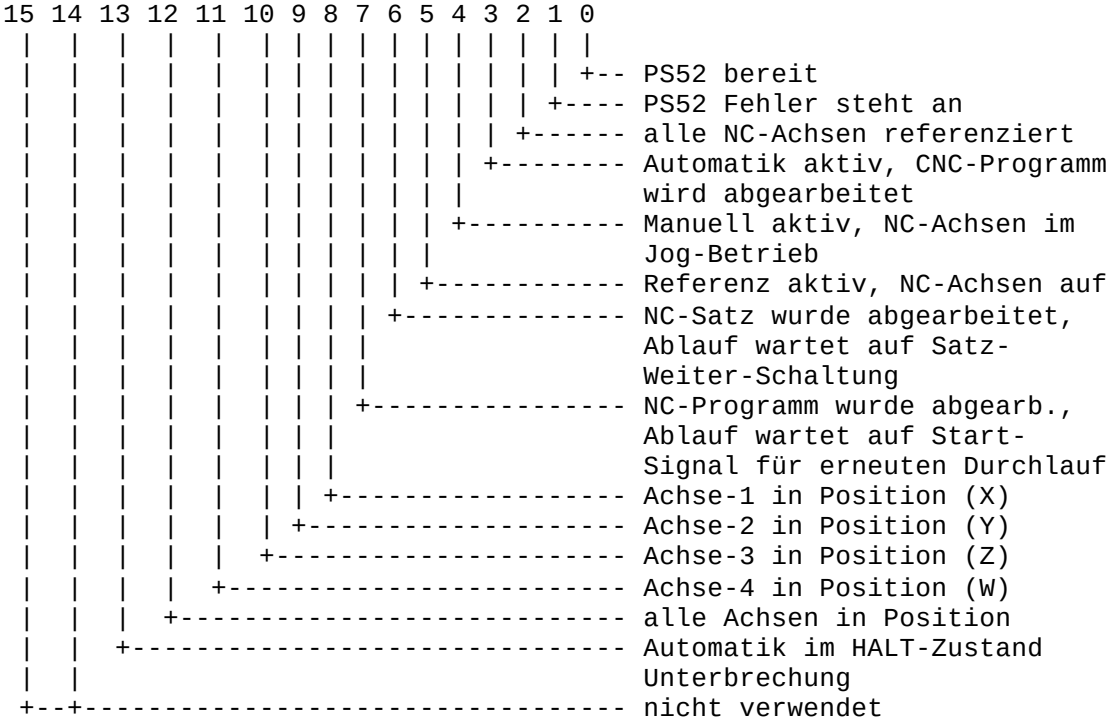
Input-Variablen:

Output-Variablen:

alActPos: ARRAY [1..4] OF DINT
Das Variablenfeld *alActPos* enthält die aktuelle Istposition der Achsen 1-4 (1-Achse(X), 2-Achse(Y)) und ist vom Typ vorzeichenbehafteter 32-Bit Wert (DINT). Die Dimension des Positionsistwertes ist 1/1000mm. Der Wert 1234700 ist also als 1234.7 mm zu interpretieren.

alNomPos: ARRAY [1..4] OF DINT
Das Variablenfeld *alNomPos* enthält die aktuelle Sollposition der Achsen 1-4 (1-Achse(X), 2-Achse(Y)) und ist vom Typ vorzeichenbehafteter 32-Bit Wert (DINT). Die Dimension des Positionssollwertes ist 1/1000mm.

uiStateOfCNC1: WORD
Die Variable *uiStateOfCNC1* ist vom Typ 16-Bit Wert und enthält interne Steuerungszustände in bit-codierter Form. Jedes Bit repräsentiert dabei einen bestimmten Steuerungszustand.



uiStateOfCNC2: WORD

Die Variable *uiStateOfCNC2* ist vom Typ 16-Bit Wert und enthält interne Steuerungszustände in bit-codierter Form. Jedes Bit repräsentiert dabei einen bestimmten Steuerungszustand.

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 |  |  |  |  |  | | | | | | | | | |
 |  |  |  |  |  | | | | | | | +-+-+-+-- Achse 1-4 auf Referenzfahrt
 |  |  |  |  |  | | | +-+-+-+---------- Achse 1-4 im Jog-Mode
 |  |  |  |  +--+-+-+------------------ = 0Fh Automatik läuft
 |  |  |  |                             = 01h Automatik unterbrochen
 +--+--+--+--------------------------- nicht verwendet
```

uiStateOfAxis: WORD

Die Variable *uiStateOfAxis* ist vom Typ 16-Bit Wert und enthält interne Steuerungszustände in bit-codierter Form. Jedes Bit repräsentiert dabei einen bestimmten Steuerungszustand.

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 |  |  |  |  |  | | | | | | | | | |
 |  |  |  |  |  | | | | | | | +-+-+-+-- Achse 1-4 in Position
 |  |  |  |  |  | | | +-+-+-+---------- Achse 1-4 ist referenziert
 |  |  |  |  +--+-+-+------------------ Achse 1-4 Interpolation aktiv
 +--+--+--+--------------------------- Achse 1-4 Lageregelung aktiv
```

uiErrorCode: WORD

Die Variable *uiErrorCode* ist vom Typ 16-Bit Wert und enthält die Nummer des aktuell anstehenden Fehlers in der PS52. Die folgenden Fehlercodes, sind Fehler die im Betriebssystem der PS52 generiert wurden. Es gibt auch die Möglichkeit Fehler aus dem SPS-Programm mit eigenen Texten zu erzeugen.

```
 0 = kein Fehler
 1 = F01 Achsen nicht referenziert
 2 = F02 Servo-Fehler X-Achse
 3 = F03 Schleppfehler X-Achse
 4 = F04 Servo-Fehler Y-Achse
 5 = F05 Schleppfehler Y-Achse
 6 = F06 Endschalter X-Achse (1)
 7 = F07 Endschalter X-Achse (2)
 8 = F08 Endschalter Y-Achse (1)
 9 = F09 Endschalter Y-Achse (2)
10 = F10 Software Endschalter X-Achse(+)
11 = F11 Software Endschalter X-Achse(-)
12 = F12 Software Endschalter Y-Achse(+)
13 = F13 Software Endschalter Y-Achse(-)
14 = F14 NC-Programmspeicher initialisiert
15 = F15 CNC-Parameterspeicher initialisiert
16 = F16 Fehler DINcode-Interpretation
17 = F17 NC-Programmspeicher voll
18 = F18 Bahngeschwindigkeit zu gross
19 = F19 Endschalter vor Referenzschalter
20 = F20 CoDeSys-Fehler
21 = F21 Fehler Pruefsumme
22 = F22 Existiert nicht!
23 = F23 Schrittmotor Einstellung!
24 = F24 Saege Ueberlauf!
```

uiPnr: WORD

Die Variable *uiPnr* ist vom Typ 16-Bit Wert und enthält die Nummer des aktuell angewählten Programms in der PS52. Der Bereich ist 0..98. Die PS52 kann 99 CNC-Programme verwalten.

```
 0 = Prg.Nr.1
24 = Prg.Nr.25
98 = Prg.Nr.99
```

uiSnr: WORD
Die Variable *uiSnr* ist vom Typ 16-Bit Wert und enthält die Nummer des aktuellen NC-Satzes im angewählten Programms in der PS52. Der Bereich ist 0..1999. Die PS52 kann 2000 CNC-Sätze verwalten und auf 99 Programme aufteilen.

```
   0 = Satz.Nr.1
  24 = Satz.Nr.25
1999 = Satz.Nr.2000
```

uiNumOfSnr: WORD
Die Variable *uiNumSnr* ist vom Typ 16-Bit Wert und enthält die Anzahl der NC-Sätze des aktuell angewählten NC-Programms mit der Prg.Nr. *uiPnr*.

```
   0 = Satz.Nr.1
  24 = Satz.Nr.25
1999 = Satz.Nr.2000
```

ulCycleCount: DINT
Die Variable *ulCycleCount* ist vom Typ 32-Bit Wert und enthält die Anzahl der Programmdurchläufe in der Betriebsart Automatik.

Beispiel in STRUCTURED TEXT :

```
(* Aufruf der Funktion *)
GetStatus();

(* Aktuelle Istposition Achsen *)
AktPositionX:= GetStatus.alActPos[1];
AktPositionY:= GetStatus.alActPos[2];

(* Abfrage, ob alle Achsen referenziert, Bit 2 *)
IF (GetStatus.uiStateOfCNC1 AND 16#0004)=16#0004 THEN
   ; (* irgendeine Aktion *)
END_IF

(* Abfrage, ob Achse 1 und 2 in Position, Bit 0/1 *)
IF (GetStatus.uiStateOfAxis AND 16#0003)=16#0003 THEN
   ; (* irgendeine Aktion *)
END_IF

(* Fehlerbehandlung *)
IF (GetStatus.uiErrorCode <> 0 THEN
   IF uiKeyCode=KEY_CLR THEN  (* CLR-Taste gedrueckt *)
      QuitError();            (* Fehler quittieren *)
   END_IF
END_IF

(* Anzeige der Programm-, Satznummer und Anzahl NC-Sätze *)
ShowValue (1, 1, 2, 0, 0, GetStatus.uiPnr+1);
ShowValue (1, 2, 2, 0, 0, GetStatus.uiSnr+1);
ShowValue (1, 3, 2, 0, 0, GetStatus.uiNumOfSnr);
```

Bemerkungen:
Die Funktion *GetStatus()* muss zyklisch im SPS-Programm aufgerufen werden, damit die Daten immer aktualisiert werden. Es ist zu beachten, dass beim ersten Aufruf keine gültigen Daten vorliegen, erst beim 2 Zyklus ist dies gewährleistet. Dies kann systembedingt nicht anders realisiert werden, da zunächst intern eine Anfrage gestartet wird und erst im nächsten Zyklus die Daten vorliegen.
Die Variablen *uiPnr* und *uiSnr* werden im Betriebssystem bei 0 beginnend verwaltet. Im SPS-Programm oder zum Anzeigen sollte immer +1 addiert werden, damit die Nummerierung bei 1 beginnt.
Mit der Funktion *SetProgIndex()* kann *uiPnr* und *uiSnr* beeinflusst/verändert werden.

4.7 JogAxis, NC-Achsen im Jog-Modus bewegen

JogAxis

Beschreibung:

Mit der Funktion *JogAxis()* besteht die Möglichkeit, eine einzelne Achse im Jog-Betrieb zu verfahren. Jog-Betrieb bedeutet, dass die Achse mit ihrer Schleich- oder Eilganggeschwindigkeit bewegt wird, solange eine der Jog-Tasten auf der PS52-Tastatur betätigt ist. Ein Loslassen dieser Tasten, führt zum Anhalten der Achse mit der parametrierten Bremsrampe. Die Jog-Tasten auf der PS52-Tastatur sind die grau/grün hinterlegten LINKS-, RECHTS-, EILGANG-Tasten.

Die Funktion *JogAxis()* ist mit diesen Tasten eng verknüpft. Die Funktion wird nur dann ausgeführt, wenn ein Jog-Kommando über die LINKS- oder RECHTS-Taste ausgelöst wird (siehe Beispiel). Das Anhalten übernimmt die PS52 innerhalb des Laufzeitsystem selbständig. Ist ein Jog-Kommando erfolgreich abgesetzt worden, kann durch Betätigen der EILGANG-Taste die Jog-Geschwindigkeit auf die parametrierte Eilgang-Geschwindigkeit erhöht werden; diese Auswertung erfolgt ebenfalls im Laufzeitsystem der PS52 und muss/kann nicht im PLC-Programm implementiert werden.

Input-Variablen:

uiAxis: WORD

Die Variable *uiAxis* ist vom Typ 16-Bit Wert und enthält den Code für die ausgewählte Achse.
0 : X-Achse
1 : Y-Achse
bei anderen Werten wird kein Jog-Betrieb ausgelöst

uiJogComm: WORD

Die Variable *uiJogComm* ist vom Typ 16-Bit Wert und enthält den Code für die ausgewählte Betriebsart, Jog-Betrieb in Richtung (-) oder in Richtung (+).
0 : kein Jog-Betrieb
1 : Jog-Betrieb(-)
2 : Jog-Betrieb(+)
bei anderen Werten wird kein Jog-Betrieb ausgelöst

Output-Variablen:

JogAxis: BOOL

Die Funktion *JogAxis()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Initialisierung, nur einmalig bei Start PLC-Programm *)
uiAxis:= 0; (* 0:X-Achse, 1:Y-Achse *)
uiJog:= 0;  (* 0:kein Jogbetrieb, 1: Jog(-), 2: Jog(+) *)

(* Abfrage Gesamtstatus *)
GetStatus();

(* warten Achsen in Position *)
IF (GetStatus.uiStateOfCNC1 AND 16#1000)=16#1000 THEN
   (* Jog-Mode X-Achse anwerfen *)
   JogAxis(uiAxis, uiJog);
   uiJog:= 0;
   CASE uiKeyCode OF
      KEY_LE : (* Jog(-) anwerfen *)
         uiJog:= 1;

      KEY_RI : (* Jog(+) anwerfen *)
         uiJog:= 2;

      KEY_UP : (* X-Achse waehlen *)
         uiAxis:= 0;

      KEY_DN : (* Y-Achse waehlen *)
         uiAxis:= 1;
   END_CASE
END_IF

(* Achsen Stop mit Rampe, wenn STOP-Taste gedrückt *)
IF uiKeyCode=KEY_STOP THEN
   PosCommand(16#0000); (* Achsen Stop mit Rampe *)
   Programmzustand:= 2; (* nächster Programmzustand *)
END_IF
```

Bemerkung:

Zu beachten ist, dass die Funktion *`JogAxis()`* mit den Tasten `LINKS, RECHTS, EILGANG` der PS52-Tastatur verknüpft ist. Ein Jog-Betrieb unabhängig vom Zustand dieser Tasten - etwa durch Auslösung anderer bool'scher Ereignisse - kann mit dieser Funktion nicht bedient werden.

4.8 LoadNCStepDown, NC-Satz von PLC-Ebene in PS52-Speicher laden

LoadNCStepDown

Beschreibung:

Mit der Funktion *LoadNCStepDown()* besteht die Möglichkeit, die Daten eines NC-Satzes im PLC-Programm zu generieren und an das Programmverwaltungssystem der PS52 zu übertragen. Die Daten eines NC-Satzes sind in Anlehnung an die DIN66025 aufgebaut. In Verbindung mit weiteren Programmverwaltungs-Routinen ist man damit in der Lage eine komplette NC-Programmverwaltung aufzubauen.
Ein NC-Satz innerhalb eines NC-Programms wird durch eine Programmnummer (Pnr) und eine Satznummer (Snr) adressiert. Um die Satzdaten in den NC-Programmspeicher zu übertragen, muss an der betreffenden Stelle im Programmspeicher (Pnr, Snr) bereits ein NC-Satz angelegt sein. Mit der Funktion *AdminNCStep()* können NC-Sätze angelegt, gelöscht oder eingefügt werden.
Ein NC-Satz kann maximal 10 Einträge enthalten. Es können also bis zu 10 Adresszeichen mit entsprechenden Werten versehen, in einem NC-Satz enthalten sein.
Bsp. NC-Satz mit 4 Einträgen
G01 X100 Y250 F100

Input-Variablen:

uiIPnr: WORD
Die Variable *uiIPnr* ist vom Typ 16-Bit Wert und enthält die Programm-Nummer des NC-Programms, das mit dieser Funktion angesprochen werden soll. Dabei ist zu beachten, dass die PS52 intern 99 NC-Programme verwalten kann, die Programmnummer in der Variablen *uiIPnr* beginnt jedoch bei 0.
min/max Wert: 0..98
Um bspw. das NC-Programm 14 anzusprechen muss in *uiIPnr* 13 eingetragen werden.

uiISnr: WORD
Die Variable *uiISnr* ist vom Typ 16-Bit Wert und enthält die Satz-Nummer innerhalb des gewählten NC-Programms (*uiIPnr*), die mit dieser Funktion angesprochen werden soll. Dabei ist zu beachten, dass die PS52 intern 1000 NC-Sätze verwalten kann, die Satznummer in der Variablen *uiISnr* beginnt jedoch bei 0.
min/max Wert: 0..999
Um bspw. das NC-Programm 14 anzusprechen und innerhalb dieses Programms den Satz 37 muss in *uiIPnr* 13 und in *uiISnr* 36 eingetragen werden.

uiNumOfAddrChars: WORD
Die Variable *uiNumOfAddrChars* ist vom Typ 16-Bit Wert und enthält die Anzahl der Elemente oder Einträge, die der NC-Satz enthält.
min/max Wert: 1..10
um bspw. den o.a. Satz mit 4 Elementen zu übertragen, muss in *uiNumOfAddrChars* = 4 eingetragen werden.

alAddrValue: DINT ARRAY[1..10]
Die Variable *alAddrValue* ist vom Typ 32-Bit Wert, signed array[1..10] und enthält die Zahlenwerte zu jedem Eintrag in aufsteigender Form. Um diese Datenstruktur mit dem o.a. NC-Satz auszufüllen sind folgende Zuweisungen im NC-Programm erforderlich:

```
alAddrValue[1] = 1;       (* G01 *)
alAddrValue[2] = 100000;  (* X100 *)
alAddrValue[3] = 250000;  (* Y250 *)
alAddrValue[4] = 1000;    (* F100 *)
```

die Werte bei den Adresszeichen X- und Y- bezeichnen die Sollwerte der Achsen; diese werden in der Dimension [mm] angegeben. Da intern Sollwerte in 1/1000 mm verarbeitet werden, muss der Sollwert entsprechend mit Faktor 1000 multipliziert werden. Das F-Adresszeichen beschreibt die Bahngeschwindigkeit, diese wird in der Dimension [0.1 mm/s] angegeben. Der Sollwert für die Bahngeschwindigkeit muss daher mit 10 multipliziert werden.

aucAddrChar: BYTE ARRAY[1..10]
Die Variable *aucAddrChar* ist vom Typ 8-Bit Wert, array[1..10] und enthält die Adresszeichen zu jedem Eintrag in aufsteigender Form. Um diese Datenstruktur mit dem o.a. NC-Satz auszufüllen sind folgende Zuweisungen im NC-Programm erforderlich:

```
aucAddrChar[1] = 'G';     (* G01 *)
aucAddrChar[2] = 'X';     (* X100 *)
aucAddrChar[3] = 'Y';     (* Y250 *)
aucAddrChar[4] = 'F';     (* F100 *)
```

Output-Variablen:

uiOPnr: WORD
uiOSnr: WORD
Die Funktion *LoadNCStepDown()* liefert die Rückgabewerte *uiOPnr* und *uiOSnr* vom Typ 16-Bit Wert, die die Programm- und Satznummer enthalten, an die gerade ein NC-Satz übertragen wurde. Um bspw. eine ganze Sequenz von NC-Sätzen zu übertragen, kann durch Rücklesen dieser Variablen verifiziert werden, ob der betreffende Satz ordnungsgemäss übertragen wurde, indem die angegebene Programm-/Satznummer mit rückgelesenen Programm-/Satz-nummer übereinstimmt.
min/max Wert: 0..999

iLoad: INT
Die Funktion *LoadNCStepDown()* liefert als Rückgabewert einen 16-bit Wert der =1 wird, wenn das Kommando erfolgreich abgesetzt werden konnte der NC-Satz in den Programmspeicher der PS52 eingetragen werden konnte.

```
iLoad = 1  (* NC-Satz konnte im Progr.speicher eingetragen werden *)
iLoad = 0  (* NC-Satz konnte im Progr.speicher nicht eingetragen
              werden, da Programm- und/oder Satznummer ausserhalb des
              zulässigen Bereichs oder an der angeg. Prog.-/Satz-
              nummer kein NC-Satz angelegt war *)
```

Beispiel:

```
(* Gesamtstatus abfragen *)
GetStatus();

(* mit Betaetigung der ENTER-Taste soll ein NC-Satz mit 4 Elementen
   im Programm 25, im Satz 13 abgespeichert werden
   die Satzdaten: G1 G90 X125.500 F75.5
 *)

(* Zustandsmaschine *)
CASE ucState OF
   (* Tastatur-Codes auswerten *)
   STATE_EVAL_KEYS:
      IF uiKeyCode=KEY_ENTER THEN
         (* Programm-Nr. und Satz-Nr. setzen *)
         uiPnr:= 24;
         uiSnr:= 12;
         SetProgIndex(uiPnr, uiSnr, 0);
         ucState:= STATE_WAIT;
      END_IF

   (* Ueberpruefung ob NC-Satz angelegt *)
   STATE_CHECK_NCSTEP:
      IF GetStatus.uiPnr=uiPnr AND
         GetStatus.uiSnr=uiSnr AND
         GetStatus.uiNumOfSnr>uiSnr THEN
         ucState:= STATE_MAKE_NCSTEP_DATA;
      ELSE
         ucState:= STATE_EVAL_KEYS;
      END_IF
```

```
        (* NC-Satzdaten in Struktur eintragen *)
        STATE_MAKE_NCSTEP_DATA:
           LoadNCStepDown.uiNumOfAddrChars:= 4;
           LoadNCStepDown.aucAddrChar[1]:= 71; (* 'G01' *)
           LoadNCStepDown.alAddrValue[1]:= 1;
           LoadNCStepDown.aucAddrChar[2]:= 71; (* 'G90' *)
           LoadNCStepDown.alAddrValue[2]:= 90;
           LoadNCStepDown.aucAddrChar[3]:= 88; (* 'X125.500' *)
           LoadNCStepDown.alAddrValue[3]:= 125500;
           LoadNCStepDown.aucAddrChar[4]:= 70; (* 'F75.5' *)
           LoadNCStepDown.alAddrValue[4]:= 755;
           ucState:= STATE_TRANSMIT_NCSTEP;

        (* NC-Satzdaten in PS52 NC-Programmspeicher uebertragen *)
        STATE_TRANSMIT_NCSTEP:
           LoadNCStepDown(uiIPnr:=uiPnr, uiISnr:=uiSnr);
           IF LoadNCStepDown.uiOPnr=uiPnr AND
              LoadNCStepDown.uiOSnr=uiSnr AND
              LoadNCStepDown.iLoad=1 THEN
              ucState:= STATE_STORE_NCDATA_IN_FLASH;
           END_IF

        (* NC-Satzdaten "flashen" *)
        STATE_STORE_NCDATA_IN_FLASH:
           StoreInFlash(1);
           ucState:= STATE_...;

        (* Warte-Zyklus *)
        STATE_WAIT:
           ucState:= STATE_CHECK_NCSTEP;
    END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Beeinflussung bzw. Verwaltung von NC-Programmen

- *LoadNCStepUp(..)*
- *AdminNCStep(..)*
- *SetProgIndex(..)*

4.9 LoadNCStepUp, NC-Satz vom PS52-Speicher in die PLC-Ebene laden

LoadNCStepUp

Beschreibung:

Mit der Funktion *LoadNCStepUp()* besteht die Möglichkeit, die Daten eines NC-Satzes aus dem Programmverwaltungssystem der PS52 in das PLC-Programm zu laden. Die Daten eines NC-Satzes sind in Anlehnung an die DIN66025 aufgebaut. In Verbindung mit weiteren Programmverwaltungs-Routinen ist man damit in der Lage eine komplette NC-Programmverwal-tung aufzubauen.
Ein NC-Satz innerhalb eines NC-Programms wird durch eine Programmnummer (Pnr) und eine Satznummer (Snr) adressiert. Um die Satzdaten aus dem NC-Programmspeicher zu laden, muss an der betreffenden Stelle im Programmspeicher (Pnr, Snr) bereits ein NC-Satz angelegt sein. Mit der Funktion *AdminNCStep()* können NC-Sätze angelegt, gelöscht oder eingefügt werden.
Ein NC-Satz kann maximal 10 Einträge enthalten. Es können also bis zu 10 Adresszeichen mit entsprechenden Werten versehen, in einem NC-Satz enthalten sein.
Bsp. NC-Satz mit 4 Einträgen
G01 X100 Y250 F100

Input-Variablen:

uiIPnr: WORD
Die Variable *uiIPnr* ist vom Typ 16-Bit Wert und enthält die Programm-Nummer des NC-Programms, das mit dieser Funktion angesprochen werden soll. Dabei ist zu beachten, dass die PS52 intern 99 NC-Programme verwalten kann, die Programmnummer in der Variablen *uiIPnr* beginnt jedoch bei 0.
min/max Wert: 0..98
Um bspw. das NC-Programm 14 anzusprechen muss in *uiIPnr* 13 eingetragen werden.

uiISnr: WORD
Die Variable *uiISnr* ist vom Typ 16-Bit Wert und enthält die Satz-Nummer innerhalb des gewählten NC-Programms (*uiIPnr*), die mit dieser Funktion angesprochen werden soll. Dabei ist zu beachten, dass die PS52 intern 1000 NC-Sätze verwalten kann, die Satznummer in der Variablen *uiISnr* beginnt jedoch bei 0.
min/max Wert: 0..999
Um bspw. das NC-Programm 14 anzusprechen und innerhalb dieses Programms den Satz 37 muss in *uiIPnr* 13 und in *uiISnr* 36 eingetragen werden.

Output-Variablen:

uiOPnr: WORD
uiOSnr: WORD
Die Funktion *LoadNCStepUp()* liefert die Rückgabewerte *uiOPnr* und *uiOSnr* vom Typ 16-Bit Wert, die die Programm- und Satznummer enthalten, von dem gerade ein NC-Satz geladen wurde. Um bspw. eine ganze Sequenz von NC-Sätzen zu laden (bspw. ein ganzes NC-Programm), kann durch Rücklesen dieser Variablen verifiziert werden, ob der betreffende Satz ordnungsgemäss übertragen wurde, indem die angegebene Programm-/Satznummer mit rückgelesenen Programm-/Satznummer übereinstimmt.
min/max Wert: 0..999

uiNumOfAddrChars: WORD
Die Funktion *LoadNCStepUp()* liefert den Rückgabewert *uiNumOfAddrChars* ist vom Typ 16-Bit Wert und enthält die Anzahl der Elemente oder Einträge, die der NC-Satz enthält.
min/max Wert: 1..10
wird bspw. der o.a. Satz mit 4 Elementen geladen, wird in *uiNumOfAddrChars* = 4 zurückgegeben werden.

alAddrValue: DINT ARRAY[1..10]
Die Funktion *LoadNCStepUp()* liefert den Rückgabewert *alAddrValue* ist vom Typ 32-Bit Wert, signed array[1..10] und enthält die Zahlenwerte zu jedem Eintrag in aufsteigender Form. Beim Laden des o.a. NC-Satzes wird diese Datenstruktur mit folgenden Werten gefüllt sein:

```
alAddrValue[1] = 1;        (* G01 *)
```

```
alAddrValue[2] = 100000;  (* X100 *)
alAddrValue[3] = 250000;  (* Y250 *)
alAddrValue[4] = 1000;    (* F100 *)
```

die Werte bei den Adresszeichen X- und Y- bezeichnen die Sollwerte der Achsen; diese werden in der Dimension [mm] angegeben. Da intern Sollwerte in 1/1000 mm verarbeitet werden, muss der Sollwert entsprechend mit Faktor 1000 multipliziert werden. Das F-Adresszeichen beschreibt die Bahngeschwindigkeit, diese wird in der Dimension [0.1 mm/s] angegeben. Der Sollwert für die Bahngeschwindigkeit muss daher mit 10 multipliziert werden.

aucAddrChar: BYTE ARRAY[1..10]

Die Funktion *LoadNCStepUp()* liefert den Rückgabewert *aucAddrChar* ist vom Typ 8-Bit Wert, array[1..10] und enthält die Adresszeichen zu jedem Eintrag in aufsteigender Form. Beim Laden des o.a. NC-Satzes wird diese Datenstruktur mit folgenden Werten gefüllt sein:

```
aucAddrChar[1] = 'G';     (* G01 *)
aucAddrChar[2] = 'X';     (* X100 *)
aucAddrChar[3] = 'Y';     (* Y250 *)
aucAddrChar[4] = 'F';     (* F100 *)
```

bLoadOk: BOOL

Die Funktion *LoadNCStepDown()* liefert als Rückgabewert auch einen bool'schen Wert, der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte und der NC-Satz aus dem Programmspeicher der PS52 geladen werden konnte.

```
bLoad=TRUE  (* NC-Satz konnte vom Progr.speicher geladen werden *)
bLoad=FALSE (* NC-Satz konnte nicht vom Progr.speicher geladen
               werden, da Programm- und/oder Satznummer ausserhalb
               des zulässigen Bereichs oder an der angeg. Prg.-/Satz-
               nummer kein NC-Satz angelegt war *)
```

Beispiel:

```
(* mit Betaetigung der ENTER-Taste soll der NC-Satz im Programm 25,
   im Satz 13 geladen werden, die Satzdaten: G1 G90 X125.500 F75.5 *)

(* Zustandsmaschine *)
CASE ucState OF
   (* Tastatur-Codes auswerten *)
   STATE_EVAL_KEYS:
      IF uiKeyCode=KEY_ENTER THEN
         (* Programm-Nr. und Satz-Nr. setzen *)
         uiPnr:= 24;
         uiSnr:= 12;
         ucState:= STATE_LOAD_NCSTEP_DATA;
      END_IF

   (* NC-Satzdaten laden *)
   STATE_LOAD_NCSTEP_DATA:
      LoadNCStepUp(uiIPnr:=uiPnr, uiISnr:=uiSnr);
      IF LoadNCStepUp.bLoadOk THEN
         Addr1 = LoadStepUp.aucAddrChar[1];
         Value1 = LoadStepUp.alAddrValue[1];
         ...
         Addr10 = LoadStepUp.aucAddrChar[10];
         Value10 = LoadStepUp.alAddrValue[10];

         ucState:= STATE_...;
      END_IF
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Beeinflussung bzw. Verwaltung von NC-Programmen

- *LoadNCStepUp(..)*
- *AdminNCStep(..)*
- *SetProgIndex(..)*

4.10 PosAxis, NC-Achsen auf Sollposition mit Interpolationsart bewegen

PosAxis

Beschreibung:

Mit dieser Funktion besteht die Möglichkeit, eine oder mehrere NC-Achsen auf eine definierbare Sollposition in einer definierbaren Interpolationsart zu bewegen.
Die Funktion ist in der IEC Terminologie als FUNKTIONSBLOCK ausprogrammiert, und muss daher vor der Verwendung instanziiert werden. Der Funktionsblock stellt also nur den Typ der Funktion dar. Mit dem Funktionsblock ist man in der Lage mehrere Instanzen dieses Funktionstyps zu bilden.

Input-Variablen:

bInit: BOOL

Die Variable *bInit* ist vom Typ Bool. Da diese Funktion intern als Ablaufmaschine ausprogrammiert ist, muss zunächst eine Initialisierung erfolgen. Ist *bInit*=TRUE wird beim nächsten Aufruf einer Instanz von *PosAxis* diese Ablaufmaschine initialisiert. Dies ist mindestes einmal beim Programmstart erforderlich (siehe Beispiel). *bInit* wird nach Bearbeitung in *PosAxis* wieder =FALSE gesetzt.

bStart: BOOL

Die Variable *bStart* ist vom Typ Bool. Ist *bStart*=TRUE wird die Positionierung der NC-Achsen gestartet. *bStart* wird nach Bearbeitung in *PosAxis* wieder =FALSE gesetzt. (Siehe Beispiel).

ucSelect: BYTE

Die Variable *ucSelect* ist vom Typ Byte. In ucSelect wird bitcodiert verschlüsselt, welche Achsen für die Positionierung angezogen werden. Man kann damit erreichen, eine, mehrere oder alle Achsen gleichzeitig zu positionieren (siehe Beispiel).

```
7 6 5 4 3 2 1 0
| | | | | | | |
| | | | | | | +-- Achse 1 aktiviert (X)
| | | | | | +---- Achse 2 aktiviert (Y)
| | | | | +------ (Achse 3 aktiviert (Z) nicht bei PS52)
| | | | +-------- (Achse 4 aktiviert (W) nicht bei PS52)
+-+-+-+---------- nicht verwendet
```

lX, lY, (lZ, lW nicht PS52): DINT

Die Variablen *lX, lY, lZ, lW* sind vom Typ DINT (32-bit vorzeichenbehaftet) und enthalten die Positionssollwerte der Achsen. Der Positionssollwert hat die Dimension [1/1000mm]. Soll bspw. Y den Positionssollwert 120.0 mm erhalten ist der Wert 120000 an *lY* zu übergeben.

uiPCom: WORD

Die Variable *uiPCom* ist vom Typ 16-Bit Wert und enthält den Code für die ausgewählte Positionierart. Dabei ist Punkt-zu-Punkt Steuerungsverhalten oder lineare Interpolation möglich.
16#0006 = Punkt-zu-Punkt Steuerungsverhalten
16#0007 = lineare Interpolation

bRel: BOOL

Die Variable *bRel* ist vom Typ Bool. Ist *bRel*=TRUE werden die übergebenen Positionssollwerte der Achsen als relative Koordinaten betrachtet. Relativ bedeutet, dass von der aktuellen Istposition der jeweiligen Achse um den Positionssollwert verfahren wird. Im Gegensatz dazu ist bei *bRel*=FALSE die absolute Positionierung angewählt. Absolut bedeutet, dass die angegebenen Sollpositionen als tatsächliche Zielkoordinaten betrachtet werden.

Output-Variablen:

bInPos: BOOL

Die Variable *bInPos* ist vom Typ Bool. Ist *bInPos*=TRUE sind alle in *ucSelect* ausgewählten NC-Achsen in Position, d.h. in Ruhe, es ist keine Bewegung im Gange. Zu beachten ist, dass nach Absetzen eines Positionierkommandos, diese Variable nicht unmittelbar =FALSE gesetzt wird, da das Positionierkommando intern erst noch verarbeitet werden muss.
Grundsätzlich eignet sich diese Variable für die Auswertung ob eine Positionierung noch läuft oder die Achsen in Ruhe (Lageregelung) sind.

cOk: SINT

Die Variable *cOk* ist vom Typ 8-Bit Wert vorzeichenbehaftet und enthält den Rückgabecode nach einem abgesetzten Positionierkommando. *cOk* kann folgende Werte annehmen:

```
1 :   das Positionierkommando wurde erfolgreich abgesetzt
0 :   die Funktion ist noch in Arbeit, das Pos.-Kommando wird gerade
      verarbeitet
-1:   das Pos.-Kommando konnte nicht abgesetzt werden, weil bspw. die
      Achsen zu diesem Zeitpunkt nicht in Ruhe waren
```

Beispiel:

```
(* DEKLARATION, Instanziierung des Funktionsblock vom Type PosAxis *)
VAR
   CNCPosAxis: PosAxis;
END_VAR

(* PROGRAMM, Initialisierung des Funktionsblocks CNCPosAxis *)
(* mit bInit=TRUE wird der Funktionsblock initialisiert,
   mit bStart=FALSE wird kein Positionierkommando ausgelöst,
   ucSelect wählt bitcodiert die Achsen X- und Y- aus,
   1-maliger Aufruf für Initialisierung *)
IF Programmzustand=1 THEN
   CNCPosAxis(bInit:=TRUE, bStart:=FALSE, ucSelect:=16#03);
   Programmzustand:= 2; (* nächster Programmzustand *)
END_IF

(* Achssollwerte setzen, warten auf Start-Ereignis (Taste, o.ä.),
   die Achssollwerte werden gesetzt X=1000.000mm, Y=123.000mm,
   StartFlag bStart=TRUE setzen *)
IF Programmzustand=2 THEN
   IF StartEreignis THEN (* Taste oder dig. Eingang oder sonst was *)
      CNCPosAxis.lX:=1000000;
      CNCPosAxis.lY:=123000;
      CNCPosAxis.bStart:=TRUE;
      Programmzustand:= 3; (* nächster Programmzustand *)
   END_IF
END_IF

(* Positionierkommando absetzen *)
(* mit uiPCom=6 wird Punkt-zu-Punkt Steuerungsverhalten angewählt,
   bRel=FALSE bedeutet absolute Zielkoordinaten,
   zyklischer Aufruf, bis cOk einen entsprechenden Wert annimmt *)
IF Programmzustand=3 THEN
   CNCPosAxis(uiPCom:=16#0006, bRel:=FALSE);
   IF CNCPosAxis.cOk=1 THEN
      Programmzustand:= 4; (* nächster Programmzustand *)
   END_IF
   IF CNCPosAxis.cOk=-1 THEN
      ; (* Pos.Kommando konnte nicht abgesetzt werden *)
   END_IF
END_IF

(* warten Achsen in Position *)
(* CNCPosAxis wird solange zykisch aufgerufen, bis bInPos=TRUE *)
IF Programmzustand=4 THEN
   CNCPosAxis;
   IF CNCPosAxis.bInPos THEN
      Programmzustand:= 2; (* nächster Programmzustand *)
   END_IF
END_IF
```

Bemerkung:

Wird das Startflag *CNCPosAxis.bStart*=TRUE gesetzt, wird beim nächsten Aufruf der *PosAxis* Funktion ein Positionierkommando ausgelöst, *CNCPosAxis.bStart* wird dann automatisch wieder = FALSE gesetzt. *PosAxis* muss aber weiterhin zyklisch aufgerufen werden um die Rückgabewerte (Output-Variablen) zu aktualisieren.
Der Funktionsblock *PosAxis* verwendet intern die Funktionen *PosCommand* und *SetNomValues*, die ebenfalls Bestandteil der PS52.Lib sind.

4.11 PosCommand, Auslösen eines best. Positionier-Kommandos

PosCommand

Beschreibung:

Mit dieser Funktion besteht die Möglichkeit, ein generelles Positionier- oder Bewegungskommando abzusetzen. Das Kommando muss sich dann nicht zwangsläufig auf in der SPS generierte Achssollwerte beziehen.
Die Funktion ist in der IEC Terminologie als FUNKTION ausprogrammiert, es genügt also ein einmaliger Aufruf für das Absetzen des Kommandos.

Input-Variablen:

uiPosComm: WORD

Die Variable *uiPosComm* ist vom Typ 16-Bit Wert und enthält den Code für das ausgewählte Positionierkommando. Folgende Tabelle beschreibt die Positionierkommandos im Detail.

```
16#0000 : Stop Achsen mit Rampe, Lageregelung bleibt aktiv
16#0002 : Start aktuell angewaehltes CNC-Programm
16#0003 : Start aktuell angewaehltes CNC-Progr.im Einzelschritt-Modus
16#0006 : Start Positionierung auf Sollwerte point-2-point (G00)
16#0007 : Start Positionierung auf Sollwerte LinInterpolation (G01)
16#000E : Stop Achsen ohne Rampe mit Reglerfreigabe aus
16#0010 : Jog-Mode X-Achse Stop
16#0020 : Jog-Mode Y-Achse Stop
16#0011 : Jog-Mode X(-) Schleichgang
16#0012 : Jog-Mode X(+) Schleichgang
16#0013 : Jog-Mode X(-) Eilgang
16#0014 : Jog-Mode X(+) Eilgang
16#0021 : Jog-Mode Y(-) Schleichgang
16#0022 : Jog-Mode Y(+) Schleichgang
16#0023 : Jog-Mode Y(-) Eilgang
16#0024 : Jog-Mode Y(+) Eilgang
16#0008 : Start Referenzfahrt Sequenz
16#0018 : Start Referenzfahrt X-Achse
16#0028 : Start Referenzfahrt Y-Achse
16#0019 : Setzen Referenz X-Achse
16#0029 : Setzen Referenz Y-Achse
```

Output-Variablen:

PosCommand: BOOL

Die Funktion *PosCommand()* liefert als Rückgabewert einen boolschen Wert der *=TRUE* wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Achsen Stop mit Rampe, wenn STOP-Taste gedrückt *)
IF Programmzustand=1 THEN
   IF uiKeyCode=KEY_STOP THEN
      PosCommand(16#0000); (* Achsen Stop mit Rampe *)
      Programmzustand:= 2; (* nächster Programmzustand *)
   END_IF
END_IF
(* warten alle Achsen in Position *)
IF Programmzustand=2 THEN
   GetStatus();
   IF (GetStatus.uiStateOfCNC1 AND 16#1000)=16#1000 THEN
      PosCommand(16#0002); (* Start aktuell angew. NC-Progr. *)
      Programmzustand:= 1; (* nächster Programmzustand *)
   END_IF
END_IF
```

Bemerkung:

siehe auch weitere Funktionen zur Bewegungssteuerung…

- *SetRequest(..)*

4.12 QuitError, anstehende Fehlermeldung quittieren

QuitError

Beschreibung:

Mit der Funktion *QuitError()* können anstehende Fehlermeldungen quittiert werden. *QuitError()* hat die gleiche Wirkung wie ein Druck auf die CLR-Taste bei anstehenden Fehlermeldungen. Mit *QuitError()* ist man jedoch in der Lage die Quittierung von Fehlermeldungen auch durch andere Ereignisse als einen Tastendruck (z.B. dig. Eingangssignal, bestimmter Zustand der Maschine, o.ä.) durchzuführen.
Mit *QuitError()* stehen somit auch beliebige andere bool'sche Ereignisse ausser der Tastatur zur Fehlerquittierung zur Verfügung.
Zu beachten ist, dass bei Fehlern die aus dem PLC-Programm generiert werden, die Kopfzeile (1. Zeile im Display) durch die Fehlermeldung überschrieben wird. Nach anschließender Fehlerquittierung muss das PLC-Programm die Restaurierung der Kopfzeile vornehmen.

Input-Variablen:

Output-Variablen:

QuitError: BOOL
Die Funktion *QuitError()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Fehlermeldung ausgeben *)
IF Fehler=1 THEN
   SetError('Fehler aus PLC-Progr');
   Fehler:= 0; (* Fehlerflag ruecks. damit nicht erneut ausgegeben *)
END_IF

(* Ruecklesen der aktuell anstehenden Fehlermeldung *)
GetStatus();
IF GetStatus.uiErrorCode<>0 THEN
   (* Fehler steht an *)
   IF E01=TRUE THEN
      (* mit E01=TRUE Fehler quittieren *)
      QuitError();
   END_IF
END_IF
```

Bemerkung:

QuitError() und *SetError()* sind Funktionen zum Aufbau eines Fehlerbehandlungssystems im PLC-Programm. Zu beachten ist der Umstand, dass Kopfzeilen im Display nach Auftreten einer Fehlermeldung restauriert werden müssen.
Fehlermeldungen führen nicht automatisch zur Abschaltung von Achsen oder I/O-Einheiten. Die Abschaltung bei Auftreten einer Fehlermeldung obliegt dem PLC-Programm und muss explizit vorgenommen werden (bspw. durch *PosCommand()*).

4.13 SetCursor, Eingabecursor an Displayposition ein-/ausschalten

SetCursor

Beschreibung:

Mit der Funktion *SetCursor()* kann die Position, die Form sowie das Vorhandensein eines Eingabe- oder Bildschirmcursors beeinflusst werden. Der Cursor wird immer dann eingeschaltet, wenn bspw. eine Eingabeaufforderung an einer bestimmten Displayposition erfolgen soll. Das Einschalten des Cursors und dessen Form wird automatisch beim Aufruf der Funktion
EditValue() verwaltet. Nach Abschluß einer Eingabe kann der Cursor durch *SetCursor(0, 0, 0)* explizit wieder abgeschaltet werden.

Input-Variablen:

uiX: WORD
uiY: WORD
Die Variablen *uiX* und *uiY* sind vom Typ 16-Bit Wert und enthalten die Displaykoordinaten. An dieser Stelle soll der Cursor auf dem Bildchirm platziert werden. Das Display der PS52 hat 4 Zeilen mit je 20 Zeichen. Um den Cursor bspw. auf Zeile 3, Spalte 17 einzuschalten, erfolgt der Aufruf so…
SetCursor(17, 3, 1);

uiAttr: WORD
Die Variable *uiAttr* ist vom Typ 16-Bit Wert und enthält ein Attribut-Wort. Dieses Attribut-Wort steuert das Ein-/ausschalten sowie die Form des Cursors

```
0000 : Cursor wird ausgeschaltet, die Koordinaten uiX und uiY sind
       dabei ohne Relevanz
0001 : Cursor wird an der Stelle uiX/uiY eingeschaltet; die Form ist
       ein permanenter Unterstrich mit der Breite eines Pixels
0002 : Cursor wird an der Stelle uiX/uiY eingeschaltet; die Form ist
       ein blinkender Block mit der Breite 8 Pixel
```

Output-Variablen:

SetCursor: BOOL
Die Funktion *SetCursor()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Cursor auschalten *)
IF Programmzustand=1 THEN
   SetCursor(0, 0, 0);
   Programmzustand:= 2; (* nächster Programmzustand *)
END_IF

(* Cursor einschalten, schmaler, permanenter Unterstrich *)
IF Programmzustand=2 THEN
   SetCursor(17, 3, 1);
   Programmzustand:= 3; (* nächster Programmzustand *)
END_IF

(* Cursor einschalten, Blockcursor, blinkender Block *)
IF Programmzustand=3 THEN
   SetCursor(33, 2, 2);
   Programmzustand:= 1; (* nächster Programmzustand *)
END_IF
```

Bemerkung:

Beim Ausschalten des Cursors sind die Koordinaten uiX und uiY ohne Relevanz.
Die Funktion *EditValue()* verwaltet das Cursor-Ein-/Ausschalten selbständig. Mit dem Start der Editor-Funktion wird der Cursor automatisch eingeschaltet; nach Abschluß einer Eingabe muss der Cursor durch *SetCursor(0, 0, 0)* explizit wieder abgeschaltet werden, sofern keine weitere Eingabe erfolgen soll (siehe auch *EditValue()*).

4.14 SetError, Fehlermeldung aus der PLC-Ebene erzeugen

SetError

Beschreibung:

Mit der Funktion *SetError()* kann eine Fehlermeldung aus dem PLC-Programm erzeugt werden. Fehlermeldungen sowohl aus dem Laufzeitsystem als auch aus dem PLC-Programm werden immer in der ersten Zeile des Displays in blinkender Form dargestellt. Der bis zu diesem Zeitpunkt dargestellte Text wird dann durch die Fehlermeldung überschrieben. Der Anwender muss also im PLC-Programm für die Restaurierung des alten Textes nach Quittierung einer Fehlermeldung selbst Sorge tragen.
Fehlermeldungen können mit der *CLR-Taste* oder durch die Funktion *QuitError()* quittiert werden. Mit *QuitError()* stehen somit auch beliebige andere bool'sche Ereignisse ausser der Tastatur zur Fehlerquittierung zur Verfügung.

Input-Variablen:

sErrString: STRING(80)
Die Variable *sErrString* ist vom Typ String (max. 80 Zeichen) und enthält den Fehlerstring, der auf dem Display ausgegeben werden soll. Bei der PS52 wird der Fehlerstring immer in der ersten Zeile angezeigt und kann maximal 20 Zeichen lang sein. Längere Fehlerstrings werden abgeschnitten.
Die Fehlermeldung aus dem PLC-Programm wird genauso wie eine Fehlermeldung aus dem Laufzeitsystem behandelt. Erst nach Quittierung des Fehlers kann die Steuerung weiterbedient werden. Eine Fehlermeldung aus dem PLC-Programm führt nicht zwangsläufig zur Abschaltung etwaiger Bewegung der Achsen. Eine gegf. notwendige Abschaltung der Achs-aktivitäten muss separat durch *PosCommand()* im PLC-Programm erfolgen.

Output-Variablen:

SetError: BOOL
Die Funktion *SetError()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Fehlermeldung ausgeben *)
IF Fehler=1 THEN
   SetError('Fehler aus PLC-Progr');
   Fehler:= 0; (* Fehlerflag ruecks. damit nicht erneut ausgegeben *)
END_IF

(* Ruecklesen der aktuell anstehenden Fehlermeldung *)
GetStatus();
IF GetStatus.uiErrorCode=0 THEN
   ; (* ...kein Fehler steht an *)
ELSE
   ; (* ...Fehler steht an, Fehlerbehandlung, z.B. Abschaltung der
        Achsen, o.ä. *)
END_IF
```

Bemerkung:

Mit der Funktion *GetStatus()* kann die aktuell anstehende Fehlermeldung in Form einer Fehlernummer rückgelesen werden (siehe Beispiel).
Mit der Funktion *QuitError()* kann eine anstehende Fehlermeldung quittiert werden.
Nachstehende Tabelle zeigt die Zuordnung der Fehlercodes zu den Textmeldungen.

Tabelle der Fehlercodes :

00 : kein Fehler steht an
01 : Achsen nicht refernziert
02 : Servo-Fehler X-Achse
03 : Schlepp-Fehler X-Achse
04 : Servo-Fehler Y-Achse
05 : Schlepp-Fehler Y-Achse
06 : Endschalter X-Achse (1)
07 : Endschalter X-Achse (2)
08 : Endschalter Y-Achse (1)
09 : Endschalter Y-Achse (2)
10 : Software-Endschalter X-Achse (1)
11 : Software-Endschalter X-Achse (2)
12 : Software-Endschalter Y-Achse (1)
13 : Software-Endschalter Y-Achse (2)
14 : NC-Programmspeicher initialisiert
15 : NC-Parameterspeicher initialisiert
16 : Fehler DIN-Code Interpreter
17 : NC-Programmspeicher voll
18 : Bahngeschwindigkeit zu gross
19 : Endschalter vor Referenzschalter bei Ref.Fahrt
20 : Fehler aus CoDeSys PLC-Programm
21 : Prüfsumme Fehler
22 : Existiert nicht!
23 : Fehler Schrittmotoreinstellung
24 : Säge Überlauf
25 : reserviert
26 : reserviert
27 : reserviert
28 : reserviert

4.15 SetMarkers, Merker aus dem PLC-Programm setzen/rücksetzen

SetMarkers

Beschreibung:

Mit der Funktion *SetMarkers()* besteht die Möglichkeit, die Zustände der Merker in der PS52 zu setzen oder rückzusetzen. 24 Merker stehen in der PS52 zur Verfügung. Da Merker auch im NC-Programm beeinflussbar sind, kann damit eine Synchronisation eines PLC- und NC-Programm-Ablaufs erfolgen.
Beispielsweise kann eine bestimmte Aktion in einem NC-Programm über einen Merkerzustand an das PLC-Programm, umgekehrt kann aus dem PLC-Programm über einen Merker ein bestimmter PLC Zustand an das NC-Programm gemeldet werden. Beide Programme (PLC und NC), die normalerweise unabhängig voneinander ablaufen, kann man auf diese Art der Merkerverarbeitung in eine gegenseitige Abhängigkeit bringen.

Input-Variablen:

uiMarkerIndex: WORD
Die Variable *uiMarkerIndex* ist vom Typ WORD (16-bit) und enthält den Index auf den zu beeinflussenden Merker. Insgesamt stehen 24 Merker zur Verfügung. Um bspw. den Merker 10 zu beeinflussen, muss die Variable *uiMarkerIndex* den Wert 10 annehmen. Werte außerhalb die außerhalb dieses Bereichs [1..24] liegen werden ignoriert.

bMarkerSet: BOOL
Die Variable *bMarkerSet* ist vom Typ BOOL. Ist *bMarkerSet* =TRUE wird der durch *uiMarkerIndex* angesprochene Merker gesetzt. Ist *bMarkerSet* =FALSE wird der durch *uiMarkerIndex* angesprochene Merker rückgesetzt. Das Setzen und Rücksetzen von Merkern kann auch in einem NC-Programm erfolgen,

Output-Variablen:

SetMarkers: BOOL
Die Funktion *SetMarkers()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(*--- alle 24 Merker init./ruecksetzen *)
IF bInit THEN
   FOR ucN:=1 TO 24 DO
      SetMarkers(ucN, FALSE);
   END_FOR
   A03:= FALSE;
   bInit:= FALSE;
END_IF

(*--- alle 24 Merkerzustaende laden *)
GetMarkers;
IF GetMarkers.bOk THEN
   (* Visualisierung des Merkerzustandes-3 durch dig. Output-3 *)
   A03:= GetMarkers.abMarker[3];

   (* Merkerzustand-3 toggeln mit dig. Input-3 *)
   E03_T(CLK:=E03);
   IF E03_T.Q THEN
      SetMarkers(3, NOT GetMarkers.abMarker[3]);
   END_IF
END_IF
```

Bemerkung:

siehe auch weitere Funktionen zur Merkerverarbeitung...

- *GetMarkers(..)*

4.16 SetNomSpeeds, Geschwindigkeitssollwerte an PS52 übertragen

SetNomSpeeds

Beschreibung:

Mit der Funktion *SetNomSpeeds()* besteht die Möglichkeit, die Verfahrgeschwindigkeit einer oder mehrerer NC-Achsen zu beeinflussen. Die Verfahrgeschwindigkeiten werden beim nächsten Bewegungsbefehl herangezogen. Die übergebenen Geschwindigkeitswerte werden durch die min/max Werte in den Steuerungsparametern begrenzt.
Zu beachten ist, dass zwischen Bahn- und Achsgeschwindigkeiten unterschieden wird. Bei Übergabe einer Bahngeschwindigkeit, resultieren die Achsgeschwindigkeiten daraus. Bei der Bahngeschwindigkeit wird nur ein Parameter übergeben, bei den Achsgeschwindigkeiten wird für jede Achse ein Parameterwert übergeben. Der Bahngeschwindigkeitswert wird immer dann angezogen, wenn der folgende Bewegungsbefehl eine Bahninterpolation durchführt (also z.B. G01, G02 oder G03). Die Achsgeschwindigkeiten werden bei einem Punkt-zu-Punkt Bewegungsbefehl angezogen (also z.B. G00).

Input-Variablen:

lSpeed1_X, lSpeed2_Y, (lSpeed3_Z, lSpeed4_W nicht PS52): DINT
Die Variablen *lSpeed1_X, lSpeed2_Y, lSpeed3_Z, lSpeed4_W* sind vom Typ DINT (32-bit vorzeichenbehaftet) und enthalten die Geschwindigkeitswerte der Achsen. Der Geschwindigkeitswert hat die Dimension [1/10 mm/s]. Soll bspw. Y den Geschwindigkeitswert 120.0 mm/s erhalten ist der Wert 1200 an *lSpeed2_Y* zu übergeben.

uiSpeedSelect: WORD
Die Variable *uiSpeedSelect* ist vom Typ WORD. Ist *uiSpeedSelect* =1 wird der in *lSpeed1_X* übergebene Geschwindigkeitswert als Bahngeschwindigkeit interpretiert. Aus dieser Bahngeschwindigkeit errechnet sich die Steuerung dann die resultierenden Achsgeschwindigkeiten. Im Gegensatz dazu werden bei *uiSpeedSelect* =0 die Geschwindigkeitswerte direkt den Achsgeschwindigkeiten zugewiesen. Für jede Achse kann ein unterschiedlicher Geschwindigkeitswert zugewiesen werden.

- uiSpeedSelect = 0
  die Achsgeschwindigkeiten werden in den Parametern lSpeed1_X, lSpeed2_Y, lSpeed3_Z, lSpeed4_W übergeben, und direkt den Achsen zugewiesen.
  Wird nur bei Punkt-zu-Punkt Bewegungsart herangezogen
  (G00 oder PosCommand(16#0006))

- uiSpeedSelect = 1
  die Bahngeschwindigkeit wird im Parameter lSpeed1_X übergeben, aus dieser Geschwindigkeot berechnet sich die Steuerung die resultierenden Achsgeschwindigkeiten.
  Wird nur bei interpolierender Bewegungsart herangezogen
  (G01 oder PosCommand(16#0007))

- uiSpeedSelect = 2
  eine bestimmte Achse soll bei der Linearinterpolation eine bestimmbare Geschwindigkeit annehmen. Die Geschwindigkeiten der anderen Achsen resultieren daraus.
  Die Zuweisung erfolgt durch Setzen des gewünschten Geschwindigkeitswertes in dem der Achse zugeordneten Geschwindigkeitsparameters. Alle anderen Geschwindigkeitswerte müssen = 0 gesetzt werden.
  Bsp. SetNomSpeeds(0, 4000, 0, 0, 2) hier wird der Y-Achse der Geschwindigkeitswert zugewiesen, mit die Achse bei einer Linearinterpolationsfahrt verfahren soll, die Geschw. der anderen Achsen werden berechnet.
  Wird nur bei interpolierender Bewegungsart herangezogen
  (G01 oder PosCommand(16#0007))

Output-Variablen:

SetNomSpeeds: BOOL
Die Funktion *SetNomSpeeds()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Abfrage Gesamtstatus *)
GetStatus();
(* warten Achsen in Position *)
IF Programmzustand=1 THEN
   IF (GetStatus.uiStateOfCNC1 AND 16#1000)=16#1000 THEN
      (* Geschwindigkeitswerte an PS52 uebertragen,
         X=120.0mm/s, Y=50.0mm/s, bez. auf Punkt-zu-Punkt Bewegung *)
      SetNomSpeeds(1200, 500, 0, 0, 0);
      (* Sollwerte an PS52 uebertragen, X=100.000mm, Y=250.000mm,
         Vermassung ABSOLUT *)
      SetNomValues(100000, 250000, 0, 0, FALSE);
      (* Start, Fahren auf Sollwerte (G00 Punkt-zu-Punkt) *)
      PosCommand(16#0006);
      Programmzustand:= 2; (* nächster Programmzustand *)
   END_IF
END_IF

(* warten Achsen in Position *)
IF Programmzustand=2 THEN
   IF (GetStatus.uiStateOfCNC1 AND 16#1000)=16#1000 THEN
      (* Bahngeschwindigkeit an PS52 uebertragen,
         X=45.0mm/s bez. auf Linearinterpolations-Bewegung *)
      SetNomSpeeds(450, 0, 0, 0, 1);
      (* Sollwerte an PS52 uebertragen, X=100.000mm, Y=250.000mm,
         Vermassung ABSOLUT *)
      SetNomValues(100000, 250000, 0, 0, FALSE);
      (* Start, Fahren auf Sollwerte (G01 Linearinterpolation) *)
      PosCommand(16#0007);
      Programmzustand:= 3; (* nächster Programmzustand *)
   END_IF
END_IF

(* PLC-Programm macht hier irgendetwas anderes *)
IF Programmzustand=3 THEN
   ;
END_IF

(* Achsen Stop mit Rampe, wenn STOP-Taste gedrückt *)
IF uiKeyCode=KEY_STOP THEN
   PosCommand(16#0000); (* Achsen Stop mit Rampe *)
   Programmzustand:= 3; (* nächster Programmzustand *)
END_IF
```

Bemerkung:

siehe auch weitere Funktionen zur Bewegungssteuerung…

- *PosAxis(..)*
- *PosCommand(..)*
- *SetNomValues(..)*

4.17 SetNomValues, Achssollwerte an PS52 übertragen

SetNomValues

Beschreibung:

Mit der Funktion *SetNomValues()* besteht die Möglichkeit, die Sollposition einer oder mehrerer NC-Achsen zu beeinflussen. Die Sollwerte sind die Positionen, die beim nächsten Bewegungsbefehl angefahren werden.
Im Funktionsblock *PosAxis()* ist diese Funktion implizit schon enthalten. Darüberhinaus kann gleichzeitig noch ein Positionierkommando abgesetzt werden.

Input-Variablen:

lAx1_X, l Ax2_Y, (l Ax3_Z, l Ax4_W nicht PS52): DINT
Die Variablen *lAx1_X, lAx2_Y, lAx3_Z, lAx4_W* sind vom Typ DINT (32-bit vorzeichenbehaftet) und enthalten die Positionssollwerte der Achsen. Der Positionssollwert hat die Dimension [1/1000mm]. Soll bspw. Y den Positionssollwert 120.0 mm erhalten ist der Wert 120000 an *lAx2_Y* zu übergeben.

bRel: BOOL
Die Variable *bRel* ist vom Typ Bool. Ist *bRel*=TRUE werden die übergebenen Positionssollwerte der Achsen als relative Koordinaten betrachtet. Relativ bedeutet, dass von der aktuellen Istposition der jeweiligen Achse um den Positionssollwert verfahren wird. Im Gegensatz dazu ist bei *bRel*=FALSE die absolute Positionierung angewählt. Absolut bedeutet, dass die angegebenen Sollpositionen als tatsächliche Zielkoordinaten betrachtet werden.

Output-Variablen:

SetNomValues: BOOL
Die Funktion *SetNomValues()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Abfrage Gesamtstatus *)
GetStatus();
(* warten Achsen in Position *)
IF Programmzustand=1 THEN
   IF (GetStatus.uiStateOfCNC1 AND 16#1000)=16#1000 THEN
      (* Sollwerte an PS52 uebertragen, X=100.000mm, Y=250.000mm,
         Vermassung ABSOLUT *)
      SetNomValues(100000, 250000, 0, 0, FALSE);
      (* Start, Fahren auf Sollwerte *)
      PosCommand(16#0006);
      Programmzustand:= 2; (* nächster Programmzustand *)
   END_IF
END_IF

(* PLC-Programm macht hier irgendetwas anderes *)
IF Programmzustand=2 THEN
   ;
END_IF

(* Achsen Stop mit Rampe, wenn STOP-Taste gedrückt *)
IF uiKeyCode=KEY_STOP THEN
   PosCommand(16#0000); (* Achsen Stop mit Rampe *)
   Programmzustand:= 2; (* nächster Programmzustand *)
END_IF
```

Bemerkung:

siehe auch weitere Funktionen zur Bewegungssteuerung…
- *PosAxis(..)*
- *PosCommand(..)*

4.18 SetProgIndex, Programm- und Satznummer CNC-Programm wählen

SetProgIndex

Beschreibung:

Mit der Funktion *SetProgIndex(..)* kann die Programm-, die Satznummer eines NC-Programms sowie der Programmzyklenzähler beeinflusst werden. Die Programmverwaltung der PS52 kann 99 Programme und insgesamt 1000 NC-Sätze verwalten. Die 1000 NC-Sätze können beliebig auf 99 Programme aufgeteilt werden. Vor der Abarbeitung eines NC-Programms wird dieses durch eine Programmnummer aufgerufen.
Der Programmzyklenzähler gibt die Anzahl der Programmdurchläufe an, also wie oft ein NC-Programm nacheinander abgearbeitet werden soll.

Input-Variablen:

iPnr: WORD
Die Variable *iPnr* ist vom Typ 16-Bit Wert signed und enthält die Programm-Nummer des NC-Programms, die mit dieser Funktion angewählt werden soll. Dabei ist zu beachten, dass die PS52 intern 99 NC-Programme verwalten kann, die Programmnummer in der Variablen *iPnr* beginnt jedoch bei 0.
min/max Wert: 0..98
Um bspw. das NC-Programm 14 anzusprechen muss in *iPnr* 13 eingetragen werden.
Sonder-Funktionen: -1, -2
Wird *iPnr* = -1 bzw. -2 gesetzt, wird eine Sonderfunktion ausgelöst, die beim Suchen nach belegten NC-Programmen hilfreich ist…
iPnr= -1 : Suche nach nächstem belegten NC-Programm
iPnr= -2 : Suche nach letztem belegten NC-Programm
…die weiteren Übergabe-Variablen sind in diesem Fall ohne Relevanz.
Beispiel:
Im NC-Programmspeicher sind die Programme 4, 17, 33, 68, 77 angelegt, alle anderen Programmplätze sind frei. Die aktuelle Programmnummer ist 17. Dies kann mit der Funktion *GetStatus()* und dem Wert *GetStatus.uiPnr = 16* ermittelt werden. Nach Aufruf von…

SetProgIndex(-1, 0, 0);
… wird das nächste angelegte NC-Programm gesucht. Mit *GetStatus()* ermittelt man … *GetStatus.uiPnr = 32*

SetProgIndex(-2, 0, 0);
… wird das letzte angelegte NC-Programm gesucht. Mit *GetStatus()* ermittelt man … *GetStatus.uiPnr = 16*

SetProgIndex(-2, 0, 0);
… wird das letzte angelegte NC-Programm gesucht. Mit *GetStatus()* ermittelt man … *GetStatus.uiPnr = 3*

iSnr: WORD
Die Variable *iSnr* ist vom Typ 16-Bit Wert signed und enthält die Satz-Nummer innerhalb des gewählten NC-Programms (*iPnr*), die mit dieser Funktion angewählt werden soll. Dabei ist zu beachten, dass die PS52 intern 1000 NC-Sätze verwalten kann, die Satznummer in der Variablen *iSnr* beginnt jedoch bei 0.
min/max Wert: 0..999
Um bspw. das NC-Programm 14 anzusprechen und innerhalb dieses Programms den Satz 37 muss in *iPnr* 13 und in *iSnr* 36 eingetragen werden.

lCycles: DINT
Die Variable *lCycles* ist vom Typ 32-Bit Wert signed und enthält den Wert des Zyklenzählers. Der Zyklenzähler gibt die Anzahl der bereits abgearbeiteten bzw. noch abzuarbeitenden Programmzyklen an. Mit jedem Durchlauf des gesamten NC-Programms wird der Zyklenzähler je nach Einstellung inkrementiert bzw. decrementiert. Bei der Einstellung dekrementierender Zyklenzähler kann somit mit *lCycles* die Anzahl der Programmdurchläufe vorgegeben werden.

Output-Variablen:

SetProgIndex: BOOL

Die Funktion *SetProgIndex()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Gesamtstatus abfragen *)
GetStatus();

(* Zustandsmaschine *)
CASE ucState OF
   (* Anzeige der Progr-, Satznr, Zykl.zaehl, Anz.Saetze des Progr.*)
   STATE_SHOW_DATA:
      ShowValue(6, 2, 2, 0, 0, GetStatus.uiPnr+1);
      ShowValue(5, 3, 3, 0, 0, GetStatus.uiSnr+1);
      ShowValue(5, 4, 3, 0, 0, GetStatus.uiNumOfSnr);
      ShowValue(15, 4, 5, 0, 0, GetStatus.ulCycleCnt);
      uiPnr:= GetStatus.uiPnr;
      lCycles:= GetStatus.ulCycleCnt;
      ucState:= STATE_EVAL_KEYS;

   (* Tastatur-Codes auswerten *)
   STATE_EVAL_KEYS:
      CASE uiKeyCode OF
         KEY_ENTER: (* Auswahl eines bestimmten NC-Programm *)
            uiPnr:= 24;
            lCycles:= 100;
            SetProgIndex(uiPnr, 0, lCycles);
            ucState:= STATE_WAIT;

         KEY_PUP: (* Suche nach naechstem belegten NC-Programm *)
            SetProgIndex(-1, 0, 0);
            ucState:= STATE_WAIT;

         KEY_PDN: (* Suche nach letztem belegten NC-Programm *)
            SetProgIndex(-2, 0, 0);
            ucState:= STATE_WAIT;
      END_CASE

   (* einen Wartezyklues einlegen *)
   STATE_WAIT:
      ucState:= STATE_SHOW_DATA;
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Auswahl oder Beeinflussung der CNC-Programme…

- *AdminNCStep(..)*
- *LoadNCStepDown(..)*
- *LoadNCStepUp(..)*
- *AdminNCStep(..)*
- *StoreInFlash(..)*

4.19 SetRequest, Anforderung an eine bestimmte Funktion von PLC absetzen

SetRequest

Beschreibung:

Mit der Funktion *SetRequest(..)* kann aus dem PLC-Programm eine Anforderung an das Laufzeitsystem der PS52 abgesetzt werden. Die Anforderungen sind Kommandos, die teilweise auch durch die Funktion *PosCommand(..)* aufgerufen werden können. Das es hier 2 verschiedene Funktionen in der PS52 Library gibt, die teilweise überlappende Funktionalität aufzeigen ist historisch und in der Struktur des PS52 Laufzeitystems bedingt.
Die Codes für die Anforderungskommandos, die durch *SetRequest(..)* möglich sind, werden folgend beschrieben.

Input-Variablen:

uiRequBit: WORD

Die Variable *uiRequBit* ist vom Typ 16-Bit Wert und enthält den Anforderungscode an die PS52, der mit dieser Funktion angewählt werden soll.

```
0000 : Stop Achsen mit Rampe, Lageregelung bleibt aktiv
0001 : Start aktuell angewaehltes CNC-Programm
0002 : Start aktuell angewaehltes CNC-Progr.im Einzelschritt-Modus
0003 : Start-Taste betätigt
0004 : Start Referenzfahrt Sequenz
0005 : Start Referenzfahrt X-Achse
0006 : Start Referenzfahrt Y-Achse

0009 : Weiterschaltung nächster Satz (S-W-S)
0010 : Weiterschaltung nächstes Programm
0011 : anstehenden Fehler quittieren
```

Output-Variablen:

SetRequest: BOOL

Die Funktion *SetRequest(..)* liefert als Rückgabewert einen boolschen Wert der *=TRUE* wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Abfrage Gesamtstatus *)
GetStatus();

(* Achsen Stop mit Rampe, wenn STOP-Taste gedrückt *)
IF Programmzustand=1 THEN
   IF uiKeyCode=KEY_STOP THEN
      PosCommand(16#0002); (* NC-Programmabarbeitung starten *)
      Programmzustand:= 2; (* nächster Programmzustand *)
   END_IF
END_IF

(* warten alle Achsen in Position *)
IF Programmzustand=2 THEN
   IF (GetStatus.uiStateOfCNC1 AND 16#1000)=16#1000 THEN
      SetRequest(9); (* weiterschalten naechster NC-Satz *)
      Programmzustand:= 3; (* nächster Programmzustand *)
   END_IF
END_IF

(* naechster Programmzustand *)
IF Programmzustand=3 THEN
   ; (* irgendetwas anwendungsspezifisches machen *)
END_IF
```

Bemerkung:

Die Abarbeitungskontrolle eines NC-Programms in der PS52 kann durch Parameter so eingestellt werden, dass nach jedem Positioniersatz die Steuerung bis zu einem erneuten Start-Impuls angehalten wird (um bspw. eine Werkstückbearbeitung vorzunehmen). Dieser Start-Impuls um von einem NC-Satz zum nächsten weiterzuschalten, wird in der multitron Terminologie als Satz-Weiter-Schaltung (S-W-S) bezeichnet. Mit der Funktion xx kann so ein Weiterschalt-Impuls auch mit dem PLC-Programm erzeugt werden. Somit könnte man sich folgenden Ablauf vorstellen:

- PLC-Kommando (PosCommand(2)), startet Abarbeitung des CNC-Programms
- warten auf Achsen in Position
- Bearbeitung in dieser Position PLC-programmgesteuert
- PLC-Kommando (SetRequest(9)), weiterschalten auf nächsten Positioniersatz

siehe auch weitere Funktionen zur Bewegungssteuerung…

- *PosCommand(..)*

4.20 ShowActualPosition, Formatierte Anzeige der aktuellen Istposition

ShowActualPosition

Beschreibung:

Mit der Funktion *ShowActualPosition()* kann an einer beliebigen Position (Displaykoordinaten) auf dem Display die aktuelle Istposition der X- und/oder Y-Achse in einem bestimmten Format ausgegeben werden. Die Istpositionswerte müssen mit der Funktion *GetStatus()* permanent aktualisiert werden.

Die Ausgabe der aktuellen Istposition der Achsen wird in vielen Anwendungen gefordert. Mit dieser Funktion kann diese Aufgabe in kompakter Form bearbeitet werden.

ShowActualPosition() nutzt die Funktion *ShowValue()* .

Dabei sind folgende Dinge zu beachten:

- die Displaykoordinaten müssen sich innerhalb des zulässigen Bereichs für das Display befinden, also 4 Zeilen / 20 Zeichen. Bei Anzeige der Istposition beider Achsen ist zu beachten, das der zweite Positionsistwert in der folgenden Zeile dargestellt wird. Für die Darstellung sind in diesem Fall 2 Displayzeilen vorzusehen.
- die Stringlänge (Anzahl der Zeichen) des Zahlenwertes resultiert aus der angegebenen Formatierung. Dies ist bei der Wahl der Displaykoordinaten zu berücksichtigen. Überlange Strings werden abgeschnitten.
- da Positionsistwerte einer permanenten Änderung unterliegen, sollte die Ausgabe zyklisch erfolgen. Die Aktualisierung der Werte auf dem Display muss aber nicht mit jedem SPS-Zyklus erfolgen. Eine Affrischung mit einem Zeitraster von 100ms ist üblich. Diese Zeitsteuerung muss aber im SPS-Programm (z.B. durch Timer-On Funktion) realisiert werden
- wird das Display gleichzeitig vom Laufzeitsystem mit anderen Textstrings beschrieben, konkurrieren die Systeme und ausgegebene Zahlenwerte werden gegf. wieder überschrieben.

Input-Variablen:

bShow: BOOL

Beim Aufruf der Funktion *ShowActualPosition()* wird überprüft, ob sich die anzuzeigenden Positionsistwerte gegenüber dem letzten Aufruf/Zyklus verändert haben. Nur bei einer Veränderung erfolgt die Ausgabe auf dem Display. Mit dieser Auswertung wird nicht mit jedem Zyklus das Display beschrieben, wodurch die CPU entlastet wird.

Bei erstmaligen Ausgeben der Positionsistwerte - etwa beim Aufbau einer neuen Maske, kann durch die Variable *bShow=TRUE* die Ausgabe erzwungen werden, auch dann, wenn sich die Positionistwerte seit der letzten Abfrage nicht verändert haben. *bShow* wird in *ShowActualPosition()* nach dem Aufruf automatisch wieder *=FALSE* gesetzt.

ucSelect: BYTE

Die Variable *ucSelect* ist vom Typ 8-Bit Wert und enthält den Code für die Auswahl der Positionsistwerte. Es besteht die Möglichkeit auszuwählen, ob keine, nur der X-, nur der Y- oder beide Positionsistwerte angezeigt werden. Die Codierung ist *ucSelect* erfolgt bitweise

```
ucSelect:= 16#00        kein Positionsistwert angewählt
ucSelect:= 16#01        Pos.istwertanzeige X- Achse angewählt
ucSelect:= 16#02        Pos.istwertanzeige Y- Achse angewählt
ucSelect:= 16#03        Pos.istwertanzeige X- und Y- Achse angewählt
```

aucX[1..2]: ARRAY OF BYTE
aucY[1..2]: ARRAY OF BYTE

Die Variablen *aucX* und *aucY* sind vom Typ 8-Bit Wert, array[1..2] und enthalten die Displaykoordinaten der beiden Positionsistwerte. Der Index/Element 1 ist der X-Achse, der Index/Element 2 ist der Y-Achse zugeordnet. An den Koordinaten werden die Positionsistwerte auf dem Display ausgegeben.

aucPre[1..2]: ARRAY OF BYTE
aucPost[1..2]: ARRAY OF BYTE

Die Variablen *aucPre* und *aucPost* sind vom Typ 8-Bit Wert, array[1..2] und enthalten die Formatangabe der beiden Positionsistwerte. Der Index/Element 1 ist der X-Achse, der Index/Element 2 ist der Y-Achse zugeordnet. *aucPre* stellt dabei die Anzahl der Vorkomma-

aucPost die Anzahl der Nachkommastellen dar. Die max. Anzahl von Stellen ist 8. *aucPre* und *aucPost* addiert dürfen also keinen größeren Wert als 8 ergeben. Ist der darzustellende Istwert kleiner als sich aus der Formatangabe ergibt, werden führende Nullen mit Leerzeichen aufgefüllt. Die Positionsistwerte sind Integer-Werte, die Komma-Stelle wird entsprechend der Formatangabe an der betreffenden Stelle gesetzt.
Wichtig: Zahlen vom Type float oder real können mit dieser Funktion nicht verarbeitet werden. Siehe auch Beispiele zur Formatangabe bei der Funktion *ShowValue()* .

aucAttr[1..2]: ARRAY OF BYTE
Die Variable *aucAttr* ist vom Typ 8-Bit Wert, array[1..2] und enthält ein Attribut-Wort. Der Index/Element 1 ist der X-Achse, der Index/Element 2 ist der Y-Achse zugeordnet. Dieses Attribut-Wort hat bei der PS52 keine Relevanz; der darin übertragene Wert hat keine Auswirkung auf die Funktion. Bei anderen Displays stellt dieses Attribut eine bestimmte Art der Displayausgabe ein. Ob bspw. der String invertiert oder mit unterschiedlichen Fonts ausgegben werden soll.
Bei der PS52 wird unabhängig vom Attribut-Wort immer nur eine Art der Zahlenwertausgabe unterstützt.

alPos[1..2]: ARRAY OF DINT
Die Variable *alPos* ist vom Typ 32-Bit Wert, arry[1..2] und enthält den anzuzeigenden Positionsistwert. Der Index/Element 1 ist der X-Achse, der Index/Element 2 ist der Y-Achse zugeordnet. Es handelt sich um einen 32-Bit vorzeichenbehafteten Integer-Wert. Andere Typen können von *ShowActualPosition()* nicht verarbeitet werden.

Output-Variablen:

Beispiel:

```
(* Abfrage Gesamtstatus *)
GetStatus();

(*--- Zustandsmaschine *)
CASE ucState OF
   (* Initialisierung, Bildschirmmaske aufbauen *)
   STATE_MASK:
      ClrScreen(0);
      SetCursor(0, 0, 0); (* Cursor ausschalten *)

      (* Displaykoord./Format der Istwertanzeige init. *)
      ShowActualPosition.ucSelect:= 16#03; (* X-/Y- Pos.Istw. gew. *)
      ShowActualPosition.aucX[1]:= 12;     (* Koord.Pos.Istw. *)
      ShowActualPosition.aucY[1]:= 2;
      ShowActualPosition.aucX[2]:= 12;
      ShowActualPosition.aucY[2]:= 3;
      ShowActualPosition.aucPre[1]:= 4;    (* Format Pos.Istw. *)
      ShowActualPosition.aucPost[1]:= 3;
      ShowActualPosition.aucPre[2]:= 4;
      ShowActualPosition.aucPost[2]:= 3;
      ShowActualPosition.aucAttr[1]:= 0;   (* Attribut Pos.Istw. *)
      ShowActualPosition.aucAttr[2]:= 0;
      ShowActualPosition.bShow:= TRUE;     (* erzwungene Anzeige *)
      ucState:= STATE_SHOW_ACTUAL_POSITION;

   (* Anzeige Positionsistwerte, zyklisch aufrufen *)
   STATE_SHOW_ACTUAL_POSITION:
      ShowActualPosition.alPos[1]:= GetStatus.alActPos[1];
      ShowActualPosition.alPos[2]:= GetStatus.alActPos[2];
      ShowActualPosition();
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Bildschirmbeeinflussung…

- *ShowValue(..)*

4.21 ShowString, Ausgabe eines Textstrings auf dem Display

ShowString

Beschreibung:

Mit der Funktion *ShowString()* kann an einer beliebigen Position (Displaykoordinaten) auf dem Display ein Textstring (ASCII-String) ausgegeben werden. Dabei sind folgende Dinge zu beachten:

- die Displaykoordinaten müssen sich innerhalb des zulässigen Bereichs für das Display befinden, also 4 Zeilen / 20 Zeichen
- die Stringlänge (Anzahl der Zeichen) kann maximal 80 Zeichen betragen. Bei der PS52 können aber nur 20 Zeichen ausgegeben werden. Überlange Strings werden abgeschnitten.
- die Ausgabe von Textstrings sollte gezielt und somit nur falls erforderlich zyklisch erfolgen, da Ausgaben auf dem Display dem Prozessor Rechenleistung für andere Aufgaben entziehen.
- wird das Display gleichzeitig vom Laufzeitsystem mit anderen Textstrings beschrieben, konkurrieren die Systeme und ausgegebene Strings werden gegf. wieder überschrieben.

Input-Variablen:

uiX: WORD
uiY: WORD

Die Variablen *uiX* und *uiY* sind vom Typ 16-Bit Wert und enthalten die Displaykoordinaten. An dieser Stelle wird der Textstring auf dem Bildchirm ausgegeben. Das Display der PS52 hat 4 Zeilen mit je 20 Zeichen. Um einen Text bspw. auf Zeile 3, Spalte 17 auszugeben, erfolgt der Aufruf so…
ShowString(17, 3, 10, 0,"Test-Texte");

uiNum: WORD

Die Variable *uiNum* ist vom Typ 16-Bit Wert und enthält die Anzahl der auszugebenden Zeichen des übergebenen Textstrings. Dabei kann der String durchaus länger sein; ausgegeben werden jedoch nur die in *uiNum* angegebene Anzahl von Textzeichen. Bei kürzeren Strings, wird der Rest mit 0-Zeichen (String-Terminus) aufgefüllt.

uiAttr: WORD

Die Variable *uiAttr* ist vom Typ 16-Bit Wert und enthält ein Attribut-Wort. Dieses Attribut-Wort hat bei der PS52 keine Relevanz; der darin übertragene Wert hat keine Auswirkung auf die Funktion. Bei anderen Displays stellt dieses Attribut eine bestimmte Art der Displayausgabe ein. Ob bspw. der String invertiert oder mit unterschiedlichen Fonts ausgegben werden soll. Bei der PS52 wird unabhängig vom Attribut-Wort immer nur eine Art der Textausgabe unterstützt.

sErrString: STRING(80)

Die Variable *sErrString* ist vom Typ String (max. 80 Zeichen) und enthält den Textstring, der auf dem Display ausgegeben werden soll. Bei der PS52 kann der Textstring maximal 20 Zeichen lang sein. Längere Strings werden abgeschnitten.

Output-Variablen:

ShowString: BOOL

Die Funktion *ShowString()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(* Aufruf einer im PLC-Programm generierten Bildschirmmaske *)
IF uiKeyCode=KEY_ENTER THEN  (* ENTER-Taste gedrückt ? *)
   (* Aufruf der Bildschirmmaske 26, wenn noch nicht aktiv *)
   IF ucActMenu <> 26 THEN
      CallMask(26, 0);
      ClrScreen(0); (* Display löschen *)
      (* Texte auf Bildschirm ausgeben *)
      ShowString(1, 1, 20, 0, '---<EIGENES MENU>---');
      ShowString(1, 2,  7, 0, 'Ist -X:');
      ShowString(1, 3,  7, 0, 'Soll-X:');
      ShowString(1, 4, 20, 0, 'noch Text in Zeile 4');
   END_IF
END_IF

(*--- Zustandsmaschine *)
CASE ucState OF
   1 : (* Bildschirmtexte einmalig auf das Display schreiben *)
      lSollwertX = 1234567;
      ShowString(1, 1, 20, 0, '---<EIGENES MENU>---');
      ShowString(1, 2,  7, 0, 'Ist -X:');
      ShowString(1, 3,  7, 0, 'Soll-X:');
      ShowString(1, 4, 20, 0, 'noch Text in Zeile 4');
      ucState:= 2;

   2 : (* Achsen-Sollwert einmalig auf das Display schreiben *)
      ShowValue(8, 3, 4, 3, 0, lSollwertX);
      ucState:= 3;

   3 : (* Achsen-Istwert zyklisch auf das Display schreiben*)
      ShowValue(8, 2, 4, 3, 0, GetStatus.alActPos[1]);
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Bildschirmbeeinflussung…

- *ClrScreen(..)*
- *ShowValue(..)*
- *EditValue(..)*
- *ShowActualPosition(..)*
- *SetError(..)*

4.22 ShowValue, Formatierte Ausgabe eines Zahlenwertes auf dem Display

ShowValue

Beschreibung:

Mit der Funktion *ShowValue()* kann an einer beliebigen Position (Displaykoordinaten) auf dem Display ein Zahlenwert in einem bestimmten Format ausgegeben werden. Dabei sind folgende Dinge zu beachten:

- die Displaykoordinaten müssen sich innerhalb des zulässigen Bereichs für das Display befinden, also 4 Zeilen / 20 Zeichen
- die Stringlänge (Anzahl der Zeichen) des Zahlenwertes resultiert aus der angegebenen Formatierung. Dies ist bei der Wahl der Displaykoordinaten zu berücksichtigen. Überlange Strings werden abgeschnitten.
- die Ausgabe von Zahlenwerten sollte gezielt und somit nur falls erforderlich zyklisch erfolgen, da Ausgaben auf dem Display dem Prozessor Rechenleistung für andere Aufgaben entziehen.
- wird das Display gleichzeitig vom Laufzeitsystem mit anderen Textstrings beschrieben, konkurrieren die Systeme und ausgegebene Zahlewerte werden gegf. wieder überschrieben.

Input-Variablen:

uiX: WORD
uiY: WORD

Die Variablen *uiX* und *uiY* sind vom Typ 16-Bit Wert und enthalten die Displaykoordinaten. An dieser Stelle wird der Zahlenwert auf dem Bildchirm ausgegeben. Das Display der PS52 hat 4 Zeilen mit je 20 Zeichen. Um einen Zahlenwert bspw. auf Zeile 3, Spalte 17 auszugeben, erfolgt der Aufruf so…

```
ShowValue(17, 3, 4, 0, 0, 1234);
```

uiPre: WORD
uiPost: WORD

Die Variablen *uiPre* und *uiPost* sind vom Typ 16-Bit Wert und enthalten die Formatangabe des auszugebenden Zahlenwertes. *uiPre* stellt dabei die Anzahl der Vorkomma- *uiPost* die Anzahl der Nachkommastellen dar. Die max. Anzahl von Stellen ist 8. *uiPre* und *uiPost* addiert dürfen also keinen größeren Wert als 8 ergeben. Ist die darzustellende Zahl kleiner als sich aus der Formatangabe ergibt, werden führende Nullen mit Leerzeichen aufgefüllt. Die darzustellende Zahl ist immer ein Integer-Wert, die Komma-Stelle wird entsprechend der Formatangabe an der betreffenden Stelle gesetzt.

Wichtig: Zahlen vom Type float oder real können mit dieser Funktion nicht verarbeitet werden. Hier einige Beispiele für die formatierte Zahlenwertausgabe:

```
ShowValue(17, 3, 4, 0, 0, 1234);          Ausgabe: _1234
ShowValue(17, 3, 4, 0, 0, -4);            Ausgabe: -___4
ShowValue(17, 3, 4, 3, 0, 123456);        Ausgabe: __123.456
ShowValue(17, 3, 2, 3, 0, -6789);         Ausgabe:   -_6.789
ShowValue(17, 3, 1, 3, 0, -123456);       Ausgabe:    -3.456
```

uiAttr: WORD

Die Variable *uiAttr* ist vom Typ 16-Bit Wert und enthält ein Attribut-Wort. Dieses Attribut-Wort hat bei der PS52 keine Relevanz; der darin übertragene Wert hat keine Auswirkung auf die Funktion. Bei anderen Displays stellt dieses Attribut eine bestimmte Art der Displayausgabe ein. Ob bspw. der String invertiert oder mit unterschiedlichen Fonts ausgegben werden soll. Bei der PS52 wird unabhängig vom Attribut-Wort immer nur eine Art der Zahlenwertausgabe unterstützt.

lVal: DINT

Die Variable *lVal* ist vom Typ 32-Bit Wert und enthält den anzuzeigenden Zahlenwert. Es handelt sich also um einen 32-Bit vorzeichenbehafteten Integer-Wert. Andere Typen können von *ShowValue()* nicht verrabeitet werden.

Output-Variablen:

ShowValue: BOOL

Die Funktion *ShowValue()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(*--- Zustandsmaschine *)
CASE ucState OF
   1 : (* Bildschirmtexte einmalig auf das Display schreiben *)
      lSollwertX = 1234567;
      ShowString(1, 1, 20, 0, '---<EIGENES MENU>---');
      ShowString(1, 2,  7, 0, 'Ist -X:');
      ShowString(1, 3,  7, 0, 'Soll-X:');
      ShowString(1, 4, 20, 0, 'noch Text in Zeile 4');
      ucState:= 2;

   2 : (* Achsen-Sollwert einmalig auf das Display schreiben *)
      ShowValue(8, 3, 4, 3, 0, lSollwertX);
      ucState:= 3;

   3 : (* Achsen-Istwert zyklisch auf das Display schreiben*)
      ShowValue(8, 2, 4, 3, 0, GetStatus.alActPos[1]);
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen zur Bildschirmbeeinflussung…

- *ClrScreen(..)*
- *ShowValue(..)*
- *EditValue(..)*
- *ShowActualPosition(..)*
- *SetError(..)*

4.23 StoreInFlash, Speichern Programm- oder Parameterdaten im FLASH

StoreInFlash

Beschreibung:

Mit der Funktion *StoreInFlash()* können bestimmte Daten oder Speicherbereiche im FLASH gespeichert werden. Das bedeutet, das diese Daten dann spannungsausfallsicher gespeichert bleiben. Zu beachten ist, das der Flash-Vorgang nicht zyklisch gerufen werden darf, da FLASH-Bausteine nur, wenn auch sehr hohe begrenzte Anzahl von Schreibzyklen erlauben (typisch 100.000 Schreibzyklen). Im Übrigen erfordert das "Flashen" relativ viel Zeit. Während des Flash-Vorgangs, kann der Prozessor keinerlei andere Aufgaben wahrnehmen. Das PLC-Programm sowie das Laufzeitsystem stehen dann für ca. 500ms still. Beim Aufruf einer Flash-Funktion werden automatisch etwaige Bewegungen der Achsen unterbunden.

Input-Variablen:

uiStoreCode: WORD

Die Variable *uiStoreCode* ist vom Typ 16-Bit Wert und enthält die Information, welche Daten im Flash gesichert werden sollen; folgend die Übersicht:

```
0001 : Save CNC-Programm
       der gesamte CNC-Programmspeicher wird im Flash mit Prüfsumme
       gespeichert
0002 : Save CNC-Parameters + remanent variables
       die CNC-Param. und rem. Variablen werden im Flash gespeichert
0003 : Save Pnr
       die aktuelle Programmnummer wird im Flash gespeichert
```

Output-Variablen:

StoreInFlash: BOOL

Die Funktion *StoreInFlash()* liefert als Rückgabewert einen boolschen Wert der =TRUE wird, wenn das Kommando erfolgreich abgesetzt werden konnte.

Beispiel:

```
(*--- "FLASHEN" *)
IF Programmzustand1=1 THEN
   (* CNC-Programme speichern *)
   StoreInFlash(1);
   Programmzustand1:= 2;
END_IF
IF Programmzustand1=2 THEN
   (* CNC-Parameter und remanente Variablen speichern *)
   StoreInFlash(2);
   Programmzustand1:= 3;
END_IF
IF Programmzustand1=3 THEN
   (*Programmnummer speichern *)
   StoreInFlash(3);
   Programmzustand1:= 0;
END_IF
```

Bemerkung:

- Flash-Routinen nicht zyklisch aufrufen (begrenzte Anzahl von Schreibzyklen)
- Flask-Routinen benötigen Zeit, während dieser keinerlei andere Funktionen ausgeführt werden (typisch 500ms..1000ms, abhängig von Datenmenge)

siehe auch weitere Funktionen …

- *LoadNCStepDown(..)*
- *AdminNCStep(..)*
- *SetProgIndex(..)*
- *VariableSave(..)*

mit diesen Funktionen wird der CNC-Programmspeicher bzw. die Programmnummer beeinflusst. Nach Aus-/Einschalten ohne vorher zu "Flashen", gehen die Datenänderungen im Programmspeicher verloren.

4.24 VariableLoad, Laden einer remanenten Variable aus dem FLASH in PLC

VariableLoad

Beschreibung:

Mit den Funktionen *VariableLoad()* und *VariableSave()* wird man in die Lage versetzt, im PLC-Programm definierte oder verwendete Variablen spannungsausfallsicher in einem dafür vorgesehenen Bereich des FLASH abzuspeichern und wieder zu restaurieren. Man spricht auch von remanenten Variablen. Eine Parameterverwaltung kann dadurch im PLC-Programm aufgebaut werden.
Nach Power-on der PS52 wird dieser Variablenbereich automatisch in den Arbeitsspeicher geladen. Von dort können mit der Funktion *VariableLoad()* einzelne Variablen in das PLC-Programm geladen werden.

Input-Variablen:

uiIndex: WORD
Die Variable *uiIndex* ist vom Typ 16-Bit Wert und zeigt auf eine indizierte Variable innerhalb des Speichers für remanente Variablen. 256 Variablen vom Typ DINT stehen zur Verfügung. *uiIndex* kann demnach die Werte…
1..256 …einnehmen
Variablenzugriffe mit Indizes ausserhalb dieses Bereichs werden nicht ausgefuehrt.

Output-Variablen:

lVariable: DINT
Die Funktion *VariableLoad()* liefert als Rückgabewert den Wert *lVariable*, der den Variablenwert des remanenten Speicherplatzes, auf den *uiIndex* zeigt enthält.

bOK: BOOL
Die Funktion *VariableLoad()* liefert weiterhin als Rückgabewert einen boolschen Wert *bOK* der =TRUE wird, wenn die Ladefunktion korrekt abgeschlossen wurde und die Variable im Rückgabewert *lVariable* steht.

Beispiel:

```
(* Unter-Programm: remanente Variablen laden 'LoadParam'*)
(* Deklaration *)
PROGRAM LoadParam
VAR_INPUT
   bInit:         BOOL;
   bLoad:         BOOL;
   ucLastIndex:   BYTE;
END_VAR
VAR
END_VAR

(* Code *)
IF bInit THEN
   bLoad:= FALSE;
   VariableLoad.uiIndex:= 1;
   bInit:= FALSE;
END_IF

(* Schleife zum Laden der Variablen *)
IF VariableLoad.uiIndex<=ucLastIndex THEN
   VariableLoad;
   IF VariableLoad.bOk THEN
      alParams[VariableLoad.uiIndex]:= VariableLoad.lVariable;
      VariableLoad.uiIndex:= VariableLoad.uiIndex + 1;
   END_IF
ELSE
   bLoad:= TRUE;
END_IF
```

```
(*--- Zustandsmaschine des Hauptprogramms *)
CASE ucState OF
   STATE_INIT: (* Zustand: Init *)
      LoadParam.bInit:= TRUE;
      ucState:= STATE_LOAD;

   STATE_LOAD: (* Zustand: Lade 16 remanente Variablen *)
      LoadParam(ucLastIndex:=16);
      IF LoadParam.bLoad THEN
         ucState:= STATE_NEXT;
      END_IF

   STATE_NEXT: (* Zustand: weiter mit anderen Steuerungsaufgaben *)
      ;
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen …

- *VariableSave(..)*
- *StoreInFlash(..)*

4.25 VariableSave, Speichern einer remanenten Variable aus PLC ins FLASH

VariableSave

Beschreibung:

Mit den Funktionen *VariableLoad()* und *VariableSave()* wird man in die Lage versetzt, im PLC-Programm definierte oder verwendete Variablen spannungsausfallsicher in einem dafür vorgesehenen Bereich des FLASH abzuspeichern und wieder zu restaurieren. Man spricht auch von remanenten Variablen. Eine Parameterverwaltung kann dadurch im PLC-Programm aufgebaut werden.
Nach Power-on der PS52 wird dieser Variablenbereich automatisch in den Arbeitsspeicher geladen. Im PLC-Programm angelegte und veränderte Variablen können mit der Funktion *VariableSave()* in diesen Arbeitsspeicher der PS52 zurückgeschrieben werden. Um diesen Arbeitsspeicher auch spannungsausfallsicher zu speichern, muss noch eine FLASH-Routine *StoreInFlash()* aufgerufen werden.
Für den Umgang mit FLASH-Routinen sind unbedingt die Angaben im Kapitel "StoreInFlash" zu beachten.

Input-Variablen:

uiIndex: WORD
Die Variable *uiIndex* ist vom Typ 16-Bit Wert und zeigt auf eine indizierte Variable innerhalb des Speichers für remanente Variablen. 256 Variablen vom Typ DINT stehen zur Verfügung. *uiIndex* kann demnach die Werte…
1..256 …einnehmen
Variablenzugriffe mit Indizes ausserhalb dieses Bereichs werden nicht ausgeführt.

lVariable: DINT
Die Variable *lVariable* ist vom Typ 32-Bit Wert signed und enthält den Variablenwert, des remanenten Speicherplatzes, auf den *uiIndex* zeigt. Dieser Wert wird im Arbeitsspeicher der PS52 abgelegt.

Output-Variablen:

bOK: BOOL
Die Funktion *VariableSave()* liefert als Rückgabewert einen boolschen Wert *bOK* der =TRUE wird, wenn die Speicherfunktion korrekt abgeschlossen wurde und die Variable im Arbeitsspeicher der PS52 steht.

Bemerkung:

Mit den Funktionen *VariableLoad()* und *VariableSave()* werden Variablen des PLC-Programms aus dem bzw. in den Arbeitsspeicher der PS52 geladen. Wird die PS52 in diesem Zustand ausgeschaltet, gehen alle Werte der remanenten Variablen verloren, da der Arbeitsspeicher ebenfalls flüchtig ist.
Erst mit der Ausführung einer FLASH-Funktion werden die Variablen spannungsausfallsicher im FLASH gespeichert.

Beispiel:

```
(*--- Zustandsmaschine des Hauptprogramms *)
CASE ucState OF
   STATE_EDIT_VAR: (* remanente Variable beeinflussen *)
      CASE uiKeyCode OF
         KEY_ENTER:
            alParams[uiIndex]:= xxx; (*Wert zuweisen *)
            ucState:= STATE_SAVE_VAR;

         KEY_DN: (* naechsten Index waehlen *)
            uiIndex:= uiIndex + 1;
            IF uiIndex > 16 THEN
               uiIndex:= 1;
            END_IF

         KEY_UP: (* vorhergehenden Index waehlen *)
            IF uiIndex > 1 THEN
               uiIndex:= uiIndex -1;
            ELSE
               uiIndex:= 16;
            END_IF

         KEY_ESC: (* Funktion verlassen *)
            IF bEdit THEN
               StoreInFlash(2);  (* remanente Variablen "flashen" *)
               bEdit:= FALSE;
            END_IF
      END_CASE

   STATE_SAVE_VAR: (* rem.Variable im PS52 Arbeistspeicher sichern *)
      IF VariableSave(alParams[uiIndex], uiIndex) THEN
         bEdit:= TRUE;
         ucState:= STATE_EDIT_VAR;
      END_IF
END_CASE
```

Bemerkung:

siehe auch weitere Funktionen …

- *VariableLoad(..)*
- *StoreInFlash(..)*

Achtung! bei Aufruf von FLASH-Routine *StoreInFlash(..)*

Flash-Routinen nicht zyklisch aufrufen (begrenzte Anzahl von Schreibzyklen)
Flash-Routinen benötigen Zeit, während dieser keine anderen Funktionen ausgeführt werden (typisch 500ms..1000ms, abhängig von Datenmenge)