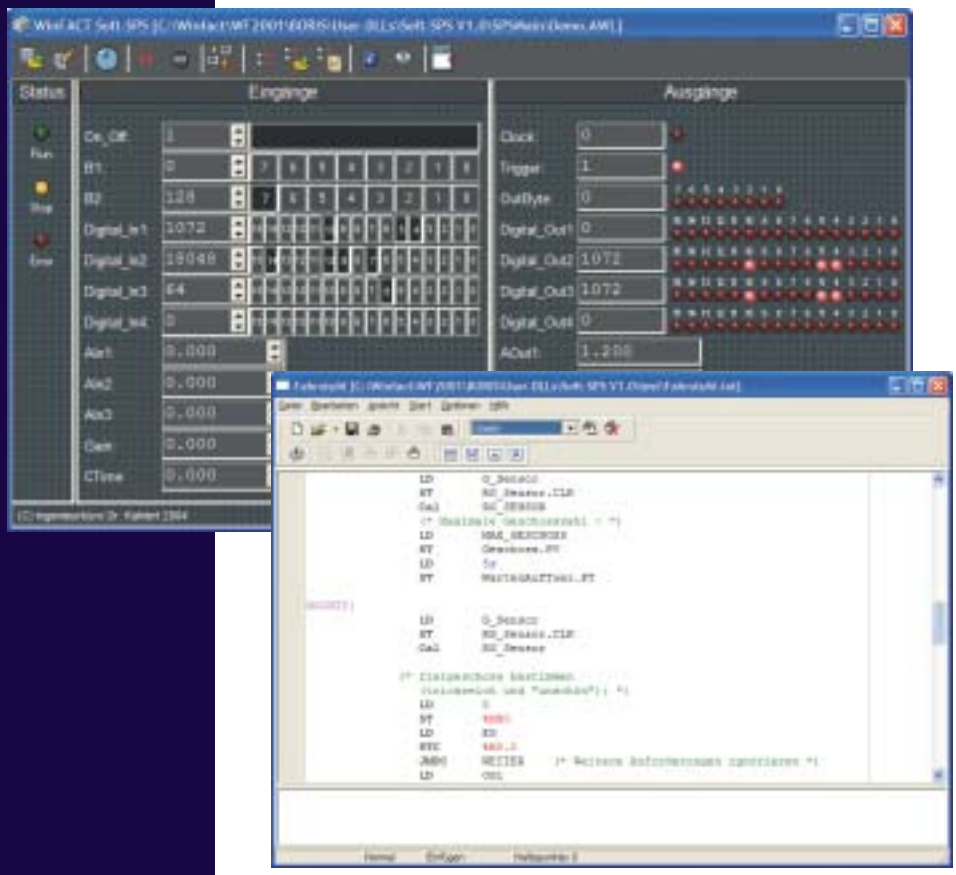
WinFACT



Soft-SPS

Benutzerhandbuch

Ingenieurbüro Dr. Kahlert

Ludwig-Erhard-Str. 45
D-59065 Hamm

Inhalt

1 Motivation

Die WinFACT-Soft-SPS ermöglicht es, schnell und komfortabel Steuerungsaufgaben zu lösen. Die Lösungen der Steuerungsaufgabe werden innerhalb der als USER-DLL programmierten Soft-SPS in Form einer Anweisungsliste erstellt. Bei dieser Tätigkeit hilft Ihnen die Entwicklungsumgebung mit einer Syntaxhervorhebung und der automatischen Codevervollständigung. Nach der Eingabe der Anweisungsliste wird der Compiler zur Übersetzung gestartet, wozu ein einfacher Knopfdruck genügt. Der Compiler arbeitet im Einschrittverfahren und erledigt seine Aufgabe sehr schnell.

Sollten Sie irgendwelche Fehler gemacht haben (nobody is perfect) so werden diese entsprechend aufgelistet. Sie können durch einen Doppelklick auf eine Fehlermeldung direkt zur entsprechenden Stelle im Quelltext springen, um die Korrektur vorzunehmen.

Ist Ihre Anweisungsliste fehlerfrei, so erhält der Block beim Schließen der Entwicklungsumgebung die entsprechenden Ein- und Ausgänge und steht für die Simulation unter BORIS bereit.[1]

Die Anweisungsliste der SPS enthält folgende Leistungsmerkmale:

- Definition von Eingangs- und Ausgangsvariablen sowie lokalen Variablen
- Einen direkt adressierbaren Merkerspeicherbereich
- Datentypen {`BOOL`, `BYTE`, `INT`, `DINT`, `WORD`, `DWORD`, `REAL`, `TIME`}
- Operatoren zum Laden und Speichern {`LD`, `LDN`, `ST`, `STN`, `STC`, `STCN`}
- Boolesche Operatoren {`SET`, `RES`, `AND`, `OR`, `XOR`, `NOT`, `ANDN`, ...} und Vergleichsoperationen {`LT`, `LE`, `EQ`, ...}
- Arithmetische Operatoren {`ABS`, `NEG`, `MUL`, `DIV`, `MOD`, `LN`, `EXP`, ...}
- Sprungfunktionen {`JMPL`, `JMP`, `JMPC`, `JMPCN`, `RET`, `RETC`, `RETCN`}
- Schiebeoperationen {`SHL`, `SHR`, `ROL`, `ROR`}
- Implizite Typkonvertierung
- Explizite Typkonvertierung
- Standardfunktionsbausteine {`TP`, `TON`, `TOF`, `CTU`, `CTD`, `CTUD`, `R_TRIG`, `F_TRIG`}
- Behandlung von Laufzeitfehlern

Bei der Suche nach logischen Fehlern können Sie den in der Entwicklungsumgebung enthaltenen Debugger verwenden. Sie können Haltepunkte definieren und schrittweise die Ausführung der Anweisungsliste verfolgen. Ein Fenster zeigt Ihnen dabei die aktuellen Werte der Variablen an, die Sie beobachten möchten.

2 Installation

Die Installation erfolgt durch das Setup-Programm *SETUP.EXE*, das sich im Wurzelverzeichnis auf dem Datenträger befindet. Das Setup-Programm kopiert die erforderlichen Dateien und Beispiele der Soft-SPS in

[1] Hinweise zur Nutzung der SPS ohne BORIS (d. h. als Stand-Alone-System) finden Sie in Kapitel 11.

das Unterverzeichnis *UserDLLs* bzw. *Examples* der WinFACT-Installation, sofern die Voreinstellungen nicht verändert werden.

Die folgenden Dateien sind fester Bestandteil der Installation:

Soft_SPS.DLL,
Soft_SPS.BMP,
Soft_SPS_T.BMP und
Soft_SPS_P.BMP.

Mit der Software werden diverse Beispiele ausgeliefert. Die folgende Tabelle bietet eine Übersicht:

Dateien	**Vom Beispiel verwendete Themen**
Förderband.BSY *Förderband.AWL*	*Einführung*, *Flankendetektoren*
CTUD.BSY *CTUD.AWL*	Zählerbausteine (s. Kapitel *Zähler*)
Timer.BSY *Timer.AWL*	Zeitgeberbausteine (s. Kapitel *Zeitgeber*)
Sprung bei Laufzeitfehler.BSY *Sprung bei Laufzeitfehler.AWL*	*Verhalten bei Laufzeitfehlern*
Sprungliste.BSY *Sprungliste.AWL*	Sprunglisten-Anweisung
Arithmetische Instruktionen.BSY *Arithmetische Instruktionen.AWL*	Arithmetische Operationen und Laufzeitfehler
Set Reset.BSY *Set Reset.AWL*	Bedingtes Setzen und Rücksetzen
Byte-Decoder.BSY *Byte-Decoder.AWL*	Verwendung des Merkerspeichers als Konvertierungsmöglichkeit (s. Kapitel *Merkerspeicher*)
Bedingter Halt.BSY *Bedingter Halt.AWL*	Demonstriert, wie ein bedingter Haltepunkt nachgebildet werden kann (Kapitel *Tipps zur Fehlersuche*).
Klammern.BSY *Klammern.AWL*	Operationen mit Klammern
Flankendetektor.BSY *Flankendetektor.AWL*	*Flankendetektoren*
Fahrstuhl.BSY *Fahrstuhl.AWL*	Zählerbausteine (s. Kapitel *Zähler*), Zeitgeberbausteine (s. Kapitel *Zeitgeber*), *Flankendetektoren*, Verwendung des Merkerspeichers als Konvertierungsmöglichkeit (s. Kapitel *Merkerspeicher*)

3 Einbinden des Soft-SPS-Blockes in eine BORIS-Struktur

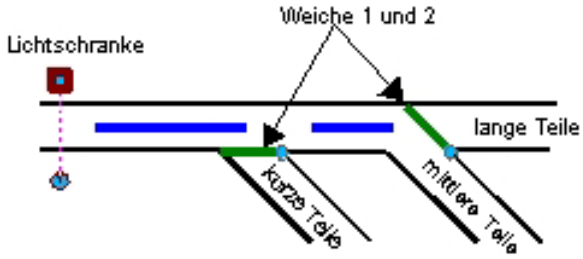
Die Soft-SPS ist als User-DLL realisiert und kann daher wie alle anderen User-DLLs genutzt werden. Zum Einfügen der Soft-SPS kann man also zunächst einen „leeren“ User-DLL-Block (Legostein-Symbol in der Systemblock-Palette *User*) einfügen und dann, nach Doppelklick auf den Block, den Dateinamen (*Soft_SPS.DLL*) auswählen. Sofern das Unterverzeichnis, in dem die Soft-SPS installiert wurde

(standardmäßig das Unterverzeichnis *\UserDLLs* Ihrer WinFACT-Installation), als Suchverzeichnis in BORIS angegeben wurde, kann der Block auch bequemer über die entsprechende Schaltfläche der Werkzeugleiste eingefügt werden (siehe nachfolgende Bildschirmgrafik).

Die Soft-SPS in der User-DLL-Systemblockpalette

4 Einführung

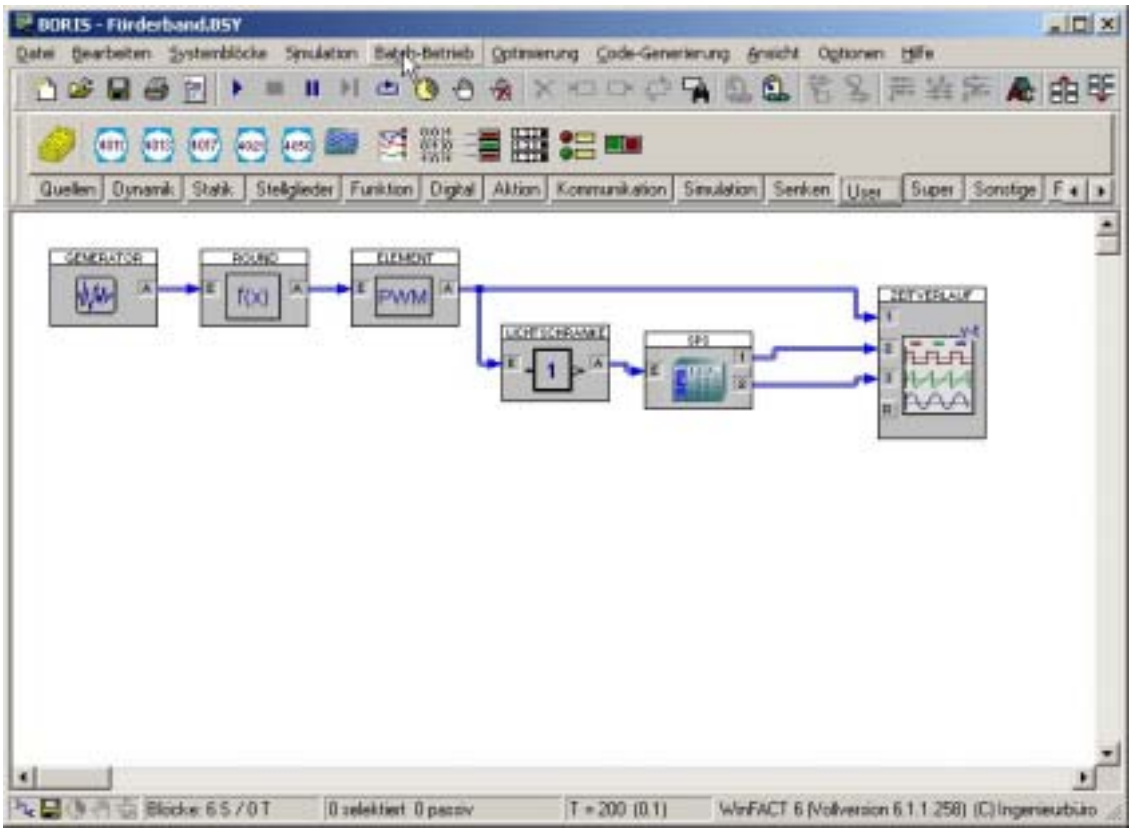
Auf einem Fließband werden drei unterschiedlich lange Teile befördert. Die Teile liegen vereinzelt auf dem Band vor und unterbrechen eine Lichtschranke unterschiedlich lang. Durch zwei Weichen sollen die Teile in die entsprechende Kiste für kurze, mittlere oder lange Teile befördert werden. Das nachfolgende Schema skizziert die Problematik:



Skizze zur Problemschilderung

Die auf dem Förderband liegenden Elemente unterbrechen die Lichtschranke für 2,5s (kurzes Element), 5s (mittleres Element) bzw. 7,5s (langes Element). Die Weiche für die mittleren Elemente (Weiche 2) soll nur gestellt werden, wenn dieses erforderlich ist. Folgt also einem mittleren Element ein kurzes, so ist es lediglich notwendig, die Weiche 1 zu stellen, Weiche 2 hingegen wird so belassen. Folgt dem kurzen jetzt ein mittleres Element, braucht nur Weiche 1 zurückgestellt zu werden, da sich Weiche 2 ja noch in der richtigen Position befindet.

Die Lösung brauchen Sie hier nicht zu entwickeln, sondern sie kann direkt als Beispiel geladen werden. Starten Sie nun BORIS und laden das Beispiel *Förderband.BSY* aus dem Beispiel-Ordner.

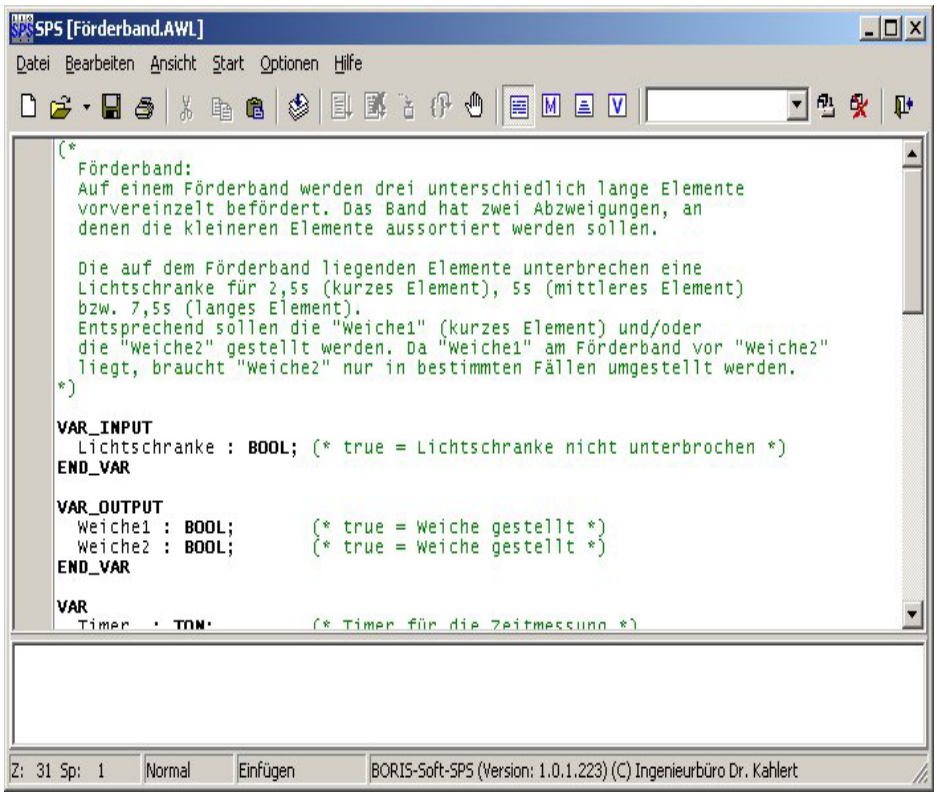


BORIS nach dem Laden des Förderband-Beispiels

Um Eingaben in der Soft-SPS vorzunehmen, führen Sie einen Doppelklick auf den Block aus und betätigen in dem erscheinenden User-DLL-Dialog die Schaltfläche DIALOG…. Es öffnet sich dann die Entwicklungsumgebung des Soft-SPS-Blockes.

Ist die Simulation aktiv, so wird bei einem Doppelklick direkt die Entwicklungsumgebung zur Fehlersuche (Debugging) geöffnet. In diesem Fall sind keinerlei Eingaben möglich (s. Kapitel Grundlagen der Entwicklungsumgebung).

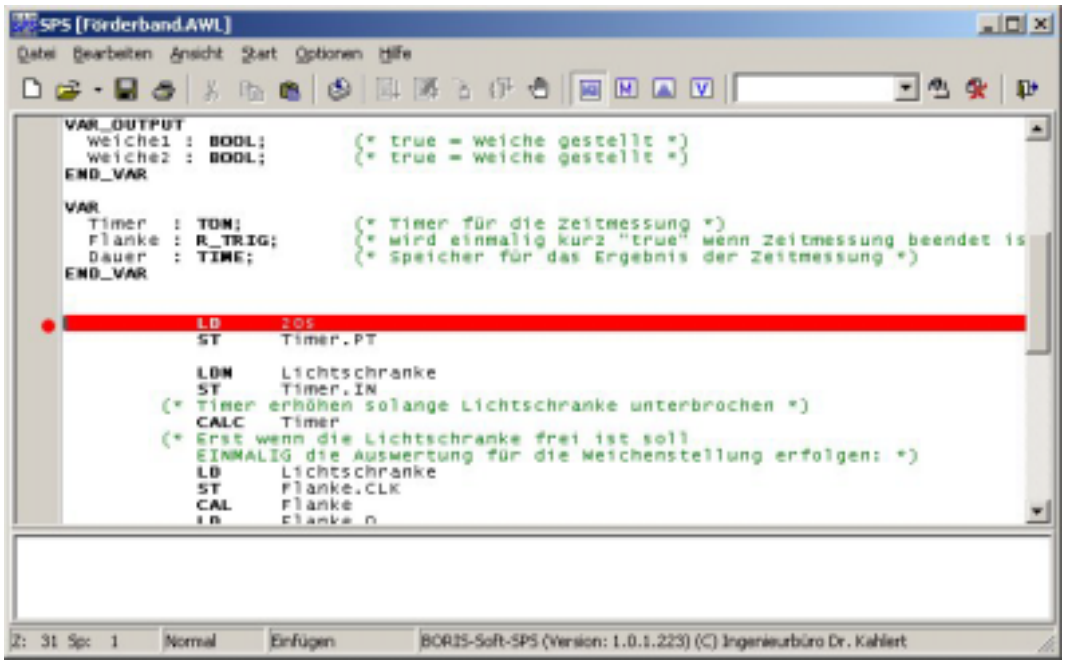
Öffnen Sie nun die Entwicklungsumgebung, ohne zuvor die Simulation zu starten.



Die Entwicklungsumgebung als Parameterdialog des SPS-Blockes

In dem Editor der Entwicklungsumgebung können Sie, wie in jedem anderen auch, Zeichen eingeben und löschen oder bestimmte Textstellen suchen oder Text ersetzen.

Es soll hier gezeigt werden, wie mit Hilfe der Entwicklungsumgebung eine AWL untersucht werden kann. Dazu soll ein Haltepunkt auf die erste Anweisung, wie im folgenden Bild dargestellt, gesetzt werden. Dies erledigen Sie, indem Sie die Schreibmarke auf die entsprechende Zeile bewegen und die Tastenkombination STRG + B betätigen (vgl. Kapitel Haltepunkte setzen und löschen).

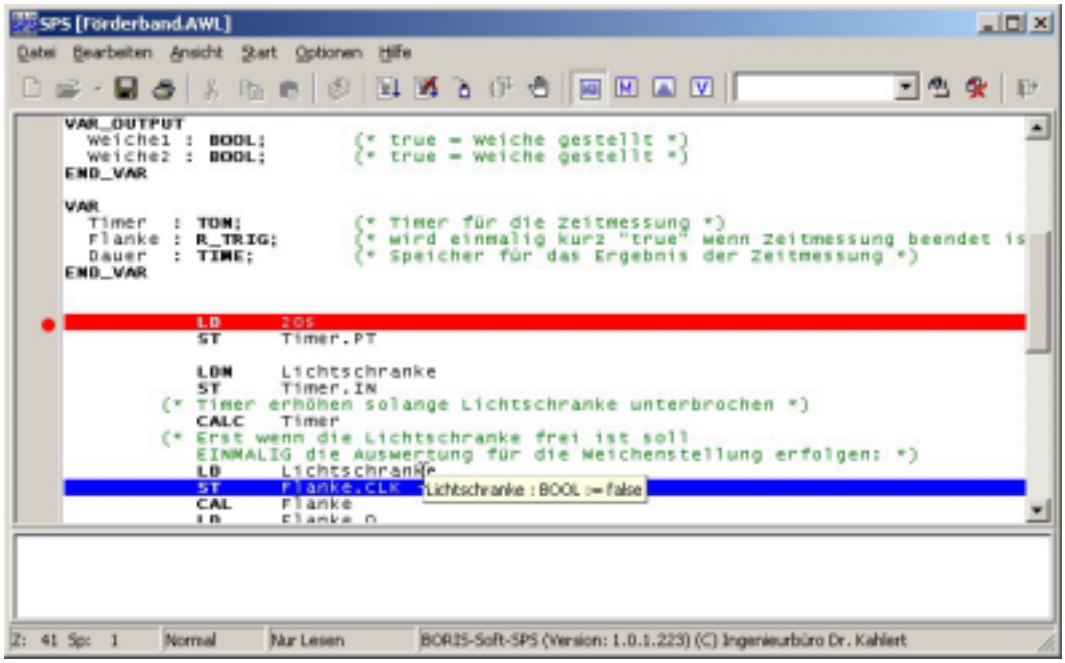


Entwicklungsumgebung mit einem Haltepunkt auf der ersten Anweisung

Schließen Sie jetzt die Entwicklungsumgebung (Menüpunkt: DATEI|SCHLIEßEN, Schaltfläche:) und den User-DLL-Dialog und starten die Simulation.

Sofort nach Simulationsstart erscheint die Entwicklungsumgebung, die nun eine Zeile mit blauem Hintergrund darstellt. In dieser Weise kennzeichnet die Soft-SPS die als nächstes auszuführende Anweisung. Die Menüpunkte AUSFÜHREN, AUSFÜHRUNG ABBRECHEN und EINZELNE ANWEISUNG aus dem Menü START und die entsprechenden Schaltflächen (, und) sind jetzt aktiv und es können nun keinerlei Veränderungen an der AWL getätigt werden (im folgenden Bild ist dies auch an dem „Nur Lesen"-Eintrag in der Statuszeile zu erkennen). Dagegen ist es nicht mehr möglich, den Dialog zu schließen, ohne dass die Ausführung der AWL beendet oder abgebrochen wurde.

Führen Sie nun die Anweisungsliste Schritt für Schritt aus, was mit der Taste F11 (oder Menüpunkt: START|EINZELNE ANWEISUNG bzw. Schaltfläche:) sehr bequem möglich ist. Nach Ausführung der ersten Anweisung erscheint die Zeile mit dem Haltepunkt wieder auf rotem Hintergrund, die aktuell auszuführende Anweisung ist wieder auf blauem Hintergrund zu sehen.



Entwicklungsumgebung nach einzelner Ausführung einiger Anweisungen

Wenn Sie während dieses Vorgangs wissen möchten, welchen Wert eine Variable hat, so können Sie den Mauszeiger auf den Namen der Variablen bewegen und dort stehen lassen. Es erscheint dann ein Hinweis mit dem Variablenbezeichner, dem Typ der Variablen und dem momentanen Wert. Im oberen Fenster ist dies für die Variable `Lichtschranke` gemacht worden. Zur Beobachtung mehrerer Variablen gibt es ein Beobachtungsfenster (s. Kapitel Variablen beobachten/verändern).

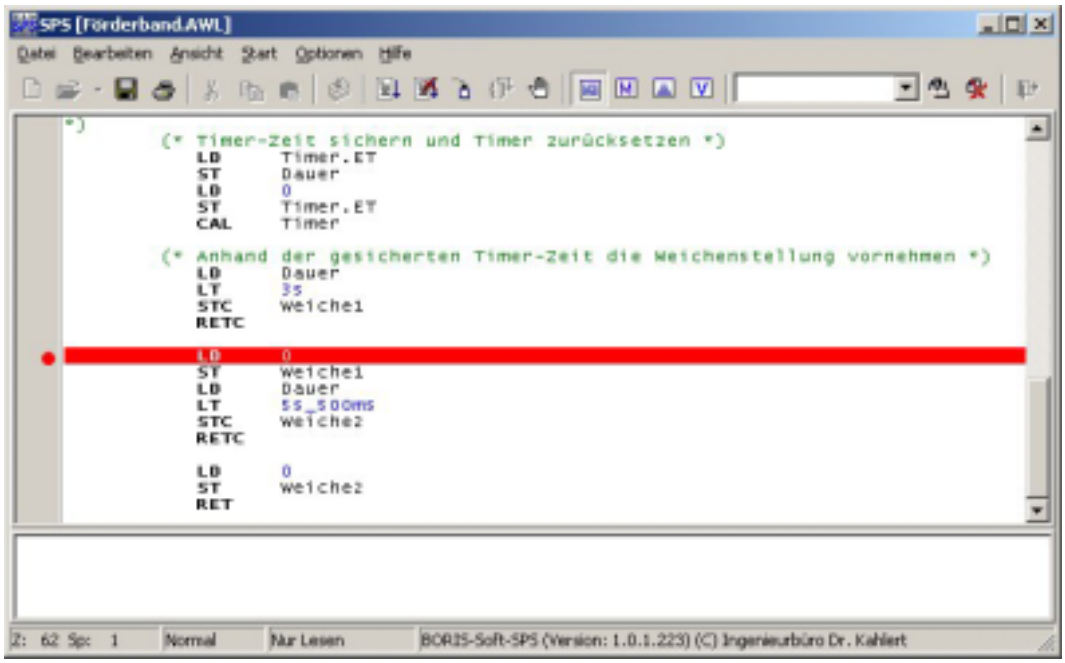
> Wenn Sie mehrere Fenster der Soft-SPS geöffnet haben (beispielsweise den Verarbeitungsstapel und das Beobachtungsfenster für Variablen), so können sie diese an dem Editierfenster „ankleben", indem Sie die Fenster langsam an den Rand des Editierfensters verschieben (s. a. Kapitel Fenster anordnen). Wenn Sie anschließend das Editierfenster verschieben, so verschieben sich automatisch alle angeklebten Fenster mit.

Sie können nun verfolgen, wie die Anweisungsliste arbeitet. Öffnen Sie zusätzlich den Verarbeitungsstapel über den Menüpunkt ANSICHT | VERARBEITUNGSSTAPEL (oder über die Schaltfläche), damit Sie die Auswirkungen der einzelnen Befehle sehen können.

Ist die Ausführung der AWL vollständig ausgeführt, so arbeitet BORIS die restlichen Blöcke für diesen Simulationsschritt ab. Anschließend wird der nächste Simulationsschritt durchgeführt und die Soft-SPS wieder am Haltepunkt angehalten. Wird das Ende der Simulation erreicht, so wird die

Entwicklungsumgebung automatisch geschlossen (s. Kapitel Grundlagen der Entwicklungsumgebung). In diesem Fall können Sie eine neue Simulation starten, um dieses Beispiel weiter zu verfolgen.

Löschen Sie nun den ersten Haltepunkt und setzen Sie einen neuen auf die AWL-Anweisung wie im folgenden Bild dargestellt. Dazu betätigen Sie jeweils die Tastenkombination Strg + B, nachdem Sie die Schreibmarke in die entsprechende Zeile bewegt haben.



Haltepunkt auf die erste Anweisung zum Rückstellen der Weiche1

Der oben gesetzte Haltepunkt befindet sich an einer Stelle, die nur dann erreicht wird, wenn die erste Weiche zurückgestellt werden soll. Sie können nun die Ausführung durch die Taste F5 (oder den Menüpunkt START|AUSFÜHREN oder die Schaltfläche) fortsetzen. Die Simulation wird nun solange ausgeführt, bis die Simulationszeit abgelaufen ist oder die Soft-SPS auf den Haltepunkt trifft.

5 Grundlagen der Entwicklungsumgebung

Die Entwicklungsumgebung arbeitet in zwei unterschiedlichen Modi, die nicht durch den Anwender eingestellt werden können, sondern automatisch aktiviert werden. Der eine ist der Normalbetrieb und wird eingeschaltet, wenn Sie den Parameterdialog eines Soft-SPS-Blockes öffnen. Sie können dann keinen weiteren Dialog öffnen, sondern müssen die Entwicklungsumgebung zunächst schließen. Hierbei wird sichergestellt, dass die durchgeführten Änderungen gespeichert werden, damit gegebenenfalls andere Soft-SPS-Blöcke, die diese AWL verwenden, ebenfalls diese Änderungen erhalten (s. Kapitel Informationen zum Laden und Speichern). Der andere Modus wird während der Simulation aktiviert. Sobald eine Soft-SPS auf einen Haltepunkt trifft oder Sie einen Doppelklick auf den SPS-Block tätigen, wird sofort die Entwicklungsumgebung im Debugbetrieb angezeigt. In diesem Modus sind keinerlei Änderungen an der AWL möglich, wodurch sichergestellt ist, dass das erzeugte Programm mit den Zeilen der AWL übereinstimmt.

Die Größe und Position der Entwicklungsumgebung und sämtlicher zugehöriger Fenster bleibt beim Wechsel der Betriebsarten unbeeinflusst.

5.1 Fenster anordnen

Die Soft-SPS enthält mehrere Fenster (zur Beobachtung von Variablen, Verarbeitungsstapel und Merkerspeicher), die frei schwebend sind und trotzdem zu einem Soft-SPS-Block gehören. Insbesondere bei der Verwendung mehrerer Soft-SPS-Blöcke kann eine frei schwebende Anordnung der zugehörigen Fenster leicht für Verwirrung sorgen. Daher ist es möglich, die zugehörigen Fenster an das Fenster der Entwicklungsumgebung „anzukleben“, indem Sie es einfach an den Rand der Entwicklungsumgebung verschieben. Anschließend wird beim Verschieben der Entwicklungsumgebung das „angeklebte“ Fenster mit verschoben. Das Aussehen oder auch die Ansicht einer Entwicklungsumgebung mit allen Fenstern wird mit dem BORIS-System gespeichert (s. Kapitel Informationen eines SPS-Blocks).

5.1.1 Speichern von Ansichten

Eine Ansicht entspricht der Anordnung (Sichtbarkeit und Position) aller Fenster eines Soft-SPS-Blockes. Diese Ansichten können in einer separaten Datei abgelegt werden. Beim Speichern der BORIS-Systemdatei wird immer die aktuelle Ansicht unabhängig von der in der Werkzeugleiste eingestellten gespeichert.

Hilfreich ist die separate Speicherung von Ansichten insbesondere bei mehreren Soft-SPS-Blöcken in einem System, die parallel untersucht werden sollen.

Zum Speichern einer Ansicht betätigen Sie den Menüpunkt ANSICHT|SPEICHERN DER MOMENTANEN SPS-ANSICHT... oder verwenden die Schaltfläche  .

5.1.2 Einstellen einer gespeicherten Ansicht

Das Einstellen einer gespeicherten Ansicht wird durch das Auswahlfeld in der Werkzeugleiste vorgenommen. In diesem werden alle verfügbaren Ansichten aufgeführt. Direkt nach dem Auswählen werden die Fenster der Soft-SPS entsprechend angeordnet.

Das Auswahlfeld zeigt nicht den augenblicklichen Zustand der Ansicht an: Werden Fenster nach einer Auswahl verschoben, so bleibt das Auswahlfeld hiervon unbeeinflusst.

5.1.3 Löschen einer Ansicht

Es kann nur die im Auswahlfeld angezeigte Ansicht gelöscht werden! Das Löschen kann über den Menüpunkt ANSICHT|LÖSCHEN DER AKTUELL EINGESTELLTEN SPS-ANSICHT... oder die Schaltfläche  erfolgen. Es wird eine Sicherheitsabfrage gestellt, damit ein versehentliches Löschen vermieden wird.

5.2 Informationen zum Laden und Speichern

Die von Ihnen getätigten Änderungen werden zur Speicherung in zwei Teile unterteilt. Zum einen sind es Informationen, die in die BORIS-Systemdatei gehören, zum anderen gibt es Dinge, die die Anweisungsliste selbst betreffen und somit an diesen Teil gebunden sind.

5.2.1 Informationen einer AWL

Die Informationen, die zu einer AWL gehören und demzufolge mit dieser abgelegt werden, sind:

- Der Quelltext der AWL,
- die Einstellungen bezüglich der Prüfungen,
- der Zykluszeitüberwachung und
- des Verhaltens beim Auftreten von Laufzeitfehlern.

5.2.2 Informationen eines SPS-Blocks

Als Informationen bezüglich eines SPS-Blockes werden angesehen:

- Sämtliche Dinge, die das Erscheinungsbild der Fenster eines Soft-SPS-Blockes betreffen,
- der Dateiname (mit relativer Pfadangabe zur Soft-SPS-DLL[2]) unter dem die AWL gespeichert wurde,
- die Haltepunkte (genauer: die Zeilen in denen ein Haltepunkt gesetzt ist), die in der AWL gesetzt worden sind und
- die Variablen, die beobachtet werden sollen.

Dadurch, dass lediglich der Dateiname in der BORIS-BSY-Datei gespeichert wird, bewirkt eine Veränderung der AWL in einem Block eine Veränderung in allen Soft-SPS-Blöcken, die diese AWL verwenden. Vor dem Öffnen der Entwicklungsumgebung und auch vor der Simulation prüfen die Soft-SPS-Blöcke, ob sich die Datei mit der AWL verändert hat und laden diese gegebenenfalls neu.

Haltepunkte besitzen zwar einen direkten Bezug zur AWL, dennoch scheint es sinnvoll zu sein, diese mit dem Soft-SPS-Block und somit mit dem BORIS-System zu speichern. Im Normalfall werden Sie Ihre AWL in einem Soft-SPS-Block in nur einem BORIS-System entwickeln. Setzen Sie diese AWL in mehreren BORIS-Systemen ein und es besteht die Notwendigkeit, die AWL zu untersuchen, so werden die Haltepunkte nicht in einem anderen System gesetzt. Weiterhin sollte bei der Verwendung mehrerer Soft-SPS-Blöcke mit dieser AWL im gleichen System nicht jeder Block die von Ihnen gesetzten Haltepunkte erhalten. Vielmehr sind Sie durch die Trennung von AWL und Haltepunkten in der Lage, in unterschiedlichen Soft-SPS-Blöcken mit der gleichen AWL unterschiedliche Haltepunkte zu setzen.

Allerdings hat diese Art der Zugehörigkeit zurzeit einen kleinen unvermeidlichen Nachteil: Die Veränderung einer AWL kann zur Folge haben, dass ein ehemals gesetzter Haltepunkt (in einem anderen Block oder in einem anderen System, der/das ebenfalls diese AWL verwendet) auf eine Zeile hinter der AWL zeigt (die AWL ist also kürzer geworden). In diesem Fall wird der Haltepunkt ohne Warnung gelöscht. Alle anderen „alten“ Haltepunkte bleiben ungeachtet der Anweisung, auf die sie nun zeigen, stehen. Es kann also durchaus sein, dass ein Haltepunkt ungültig wird, da er nun auf eine Kommentarzeile verweist.

Für Variablen gilt dementsprechend das gleiche, d. h. Sie können in unterschiedlichen Soft-SPS-Blöcken, die die gleiche AWL verwenden, unterschiedliche Variablen beobachten. Variablen, die nach einer Bearbeitung der AWL nicht mehr vorhanden sind, werden aus der Liste zu beobachtender Variablen entfernt.

5.3 Der Editor der Entwicklungsumgebung

Der AWL-Editor hat alle Eigenschaften und Fähigkeiten eines gewöhnlichen Editors. Darüber hinaus werden syntaktische Eigenheiten der AWL speziell hervorgehoben, wodurch der Quelltext verständlicher und übersichtlicher erscheint.

Für das Eingeben und Formatieren einer Anweisungsliste stehen Ihnen zwei Betriebsarten zur Verfügung, die zusätzlichen Funktionsumfang bieten:

- Zeilenauswahlmodus
- Spaltenauswahlmodus

In der nachfolgenden Tabelle sind sämtliche Tastenkombinationen aufgeführt, die ein schnelleres und effizienteres Editieren des Quelltextes erlauben. Unter anderem finden Sie hier auch die Befehle zur Umschaltung zwischen den einzelnen Betriebsarten.

[2] Im Gegensatz zur absoluten Pfadangabe enthält eine relative Pfadangabe nur Verzeichnisse, die von einem Verzeichnis zu einem anderen durchlaufen werden müssen. Hier also von dem Verzeichnis der Soft-SPS-DLL zu dem Verzeichnis, in dem sich die AWL befindet. Dabei bezeichnet „**..**“ das dem aktuellen Verzeichnis übergeordnete Verzeichnis. Der Vorteil dabei liegt in der besseren Portierbarkeit eines BORIS-Systems.

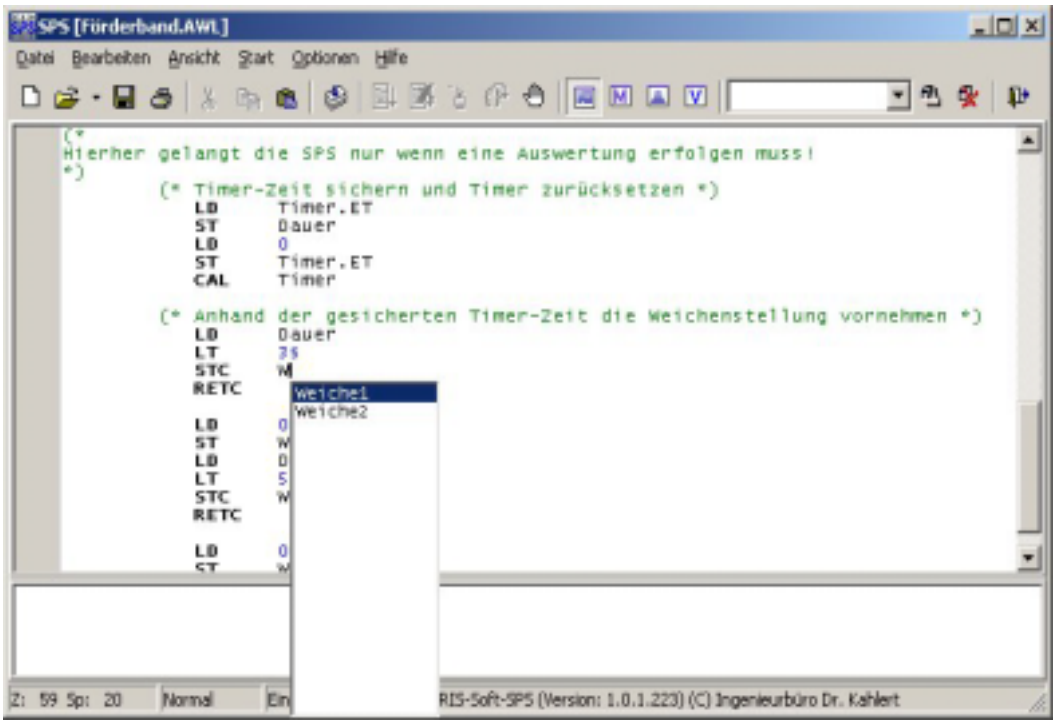
5.3.1 Tastaturbefehle des Editors

Hier sollen nun einzelne Tastaturkommandos für den Editor aufgeführt werden:

Tastenkombination	Wirkungsweise im Editor
UMSCHALT. + RECHTS UMSCHALT. + LINKS	Wählt das Zeichen rechts bzw. links von der Schreibmarke aus.
UMSCHALT. + AUF UMSCHALT. + AB	Wählt die vorangehende bzw. nächste Zeile aus.
STRG + RECHTS STRG + LINKS	Verschiebt die Schreibmarke wortweise nach links bzw. nach rechts.
STRG +UMSCHALT. + RECHTS STRG +UMSCHALT. + LINKS	Wählt das Wort rechts bzw. links von der Schreibmarke aus.
STRG + AUF STRG + AB	Verschiebt den Bildschirminhalt nach oben bzw. unten (die Schreibmarke bleibt über dem Inhalt stehen).
UMSCHALT + BILD AUF UMSCHALT + BILD AB	Wählt mehrere Zeilen (Anzahl von der Zeilenzahl des sichtbaren Bereiches abhängig) aus.
UMSCHALT + POS 1 UMSCHALT + ENDE	Wählt die Zeichen von der aktuellen Position bis zum Zeilenanfang bzw. bis zum Zeilenende aus.
STRG + POS 1 STRG + ENDE	Springt an den Anfang bzw. an das Ende des Quelltextes.
UMSCHALT + STRG + POS 1 UMSCHALT + STRG + ENDE	Wählt alle Zeichen von der aktuellen Position bis zum Anfang bzw. bis zum Ende des Quelltextes aus.
STRG + RÜCKWÄRTS	Löscht die Zeichen links neben der Schreibmarke bis zum Wortanfang.
STRG + T	Löscht die Zeichen rechts neben der Schreibmarke bis zum Wortende.
UMSCHALT + STRG + B	Bewegt die Schreibmarke zur korrespondierenden Klammer. Die Schreibmarke muss sich dazu an der entsprechenden anderen Klammer befinden.
UMSCHALT + STRG + I UMSCHALT + STRG + U	Verschiebt die ausgewählten Zeilen nach rechts bzw. links.
UMSCHALT + STRG + C	Schaltet den Editor in den Spaltenauswahlmodus.
UMSCHALT + STRG + L	Schaltet den Editor in den Zeilenauswahlmodus.
UMSCHALT + STRG + N	Schaltet den Editor in den normalen Auswahlmodus.

5.3.2 Automatische Codevervollständigung

Die automatische Codevervollständigung erlaubt Ihnen eine schnelle, korrekte und konsistente Eingabe der AWL. Sie tritt in Kraft, wenn sich die Schreibmarke an einer syntaktisch fehlerhaften Stelle im Quelltext befindet. Dazu wird ein Teil des Quelltextes analysiert, um eine möglichst gute Prognose für die Vervollständigung zu geben.

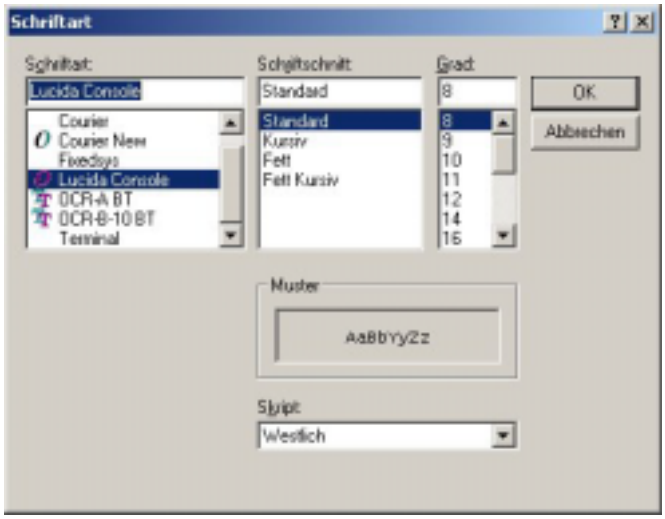


Editor mit aktiver Codevervollständigung

Die Analyse des Quelltextes erstreckt sich immer über den gesamten Bereich der Variablendeklaration und über 150 Zeilen vor und hinter der aktuellen Zeile. Daher werden durch die automatische Codevervollständigung keine Sprungmarken als Ziele für Sprunganweisungen (`JMP`, `JMPC` und `JMPCN`) vorgeschlagen, die weiter als 150 Zeilen entfernt sind - auch wenn dies syntaktisch in Ordnung wäre.

5.3.3 Verändern der Schriftart

Sie können die Schriftart des Editors verändern, indem Sie den Menüpunkt OPTIONEN|EDITOR... wählen. Es erscheint dann ein Dialog zur Auswahl der Schriftart und Schriftgröße.



Auswahl der Schriftart des Editors

In diesem Dialog können nur Schriften mit fester Zeichenbreite ausgewählt werden, die für Programmiersprachen allgemein üblich sind und aufgrund des festen Zeichenabstandes eine bessere Formatierung erlauben.

6 Programmierung in Anweisungsliste

Die Soft-SPS wird zurzeit ausschließlich in AWL programmiert. Die verwendbaren Anweisungen lehnen sich sehr stark an die der Norm IEC 61131-3 an. Dies hat den Vorteil, dass

- die Anweisungen keine Symbole mehr enthalten (wie z. B. `+I`, `-R`, etc.) und somit leichter zu lesen sind,
- Sie sich nicht um diverse Flags in einem Statusregister kümmern müssen, sondern stattdessen einfache Vergleichsanweisungen verwenden können,
- keine Datenbausteine mehr notwendig sind,
- Sie Variablen verwenden können und
- sich nicht um organisatorische Dinge (wie die Frage: Welcher Merker wird für was eingesetzt?) kümmern müssen.

Dennoch besteht weiterhin die Möglichkeit, einen bestimmten Speicherbereich direkt anzusprechen (s. u. Merkerspeicher).

6.1 Datentypen

Die Soft-SPS für *WINFACT* unterstützt die folgenden Datentypen:

Typ	**Größe**	**Wertebereich (für Erklärungen s. *Lexikalische Konstanten*)**
`BOOL`	1 Bit	true, false
`BYTE`	8 Bit	0..255
`WORD`	16 Bit	0..65535
`DWORD`	32 Bit	0..4294967296
`INT`	16 Bit	-32768..+32767
`DINT`	32 Bit	-2147483648..2147483647
`REAL`	32 Bit	-1.2E-38 .. 1.2E38; Genauigkeit (dezimal): 6 Stellen
`TIME`	32 Bit	0..49d_17h_2m_47s_295ms; Auflösung 1 ms

Die Typen `INT`, `DINT` und `REAL` sind die einzigen vorzeichenbehafteten Typen.

6.2 Lexikalische Konstanten

Lexikalische Konstanten sind Zeichenketten wie z. B. `3.14`, die wir als Mensch automatisch als einen bestimmten Wert interpretieren. Sie sind die Grundlage zur Definition einer Variablen oder Konstanten; daher ist es wichtig zu wissen, wie der Compiler solche Zeichenketten interpretiert. Genauer gesagt, welcher Datentyp bei einer lexikalischen Konstanten vom Compiler ermittelt wird. Das folgende einfache Regelwerk wird hierzu verwendet:

- Handelt es sich bei der gelesenen Zeichenkette um eine einfache Zahl, so wird diese unabhängig von ihrer Größe als ein `INT` betrachtet.
- Enthält die Zeichenkette `d`, `h`, `m`, `s` oder `ms` (nur Kleinbuchstaben), so ist der Typ `TIME`.
- Enthält die Zeichenkette einen Dezimalpunkt, so ist der Typ `REAL`.

- Entspricht die Zeichenkette den Wörtern TRUE, FALSE (ungeachtet der Groß-/Kleinschreibung), so ist der Typ BOOL.

Das obige Regelwerk ist aber nicht ausreichend, um einen WORD Typen anzugeben. Daher können alle Typen (s. o.) mit einem nachfolgendem # einer Konstantendefinition vorangestellt werden. So definieren:

- BYTE#1 den Typ BYTE mit dem Wert 1 und
- WORD#5 den Typ WORD mit dem Wert 5.

Die Angabe von BYTE#300 führt allerdings zu einem Fehler, da der Wert 300 nicht in den Typen BYTE passt. Bei den ganzzahligen Typen (alle außer BOOL, REAL und TIME) ist es manchmal wünschenswert, den Wert in binärer, oktaler oder hexadezimaler Schreibweise anzugeben. Dies kann mit Hilfe der Angabe der Basis vor der Zahl erledigt werden:

- 16#16 ist also ein INT mit dem Wert 24 (die erste 16 entspricht der Angabe der Basis)
- BYTE#2#0001_1001 ist ein BYTE mit dem Wert 25
- WORD#8#12 entspricht dem Wert 10 als Typ WORD

Konstanten vom Typ TIME und Werte im Binärformat dürfen zur besseren Lesbarkeit Unterstriche enthalten. Demzufolge definieren:

- 5d4h3m2s1ms und
- 5d_4h_3m_2s_1ms

die gleiche Zeitkonstante. Dabei bedeuten:

d	Tage,
h	Stunden,
m	Minuten,
s	Sekunden und
ms	Millisekunden.

Des Weiteren muss eine Zeitangabe nicht alle Einheiten besitzen. So ist 2d_1h_5m_30s_500ms mit 49h_5m_30s_500ms und 2945m30s500ms identisch.

6.3 Deklaration von Variablen

Variablen werden zu Beginn einer Anweisungsliste deklariert und gegebenenfalls initialisiert, d. h. auf einen bestimmten Wert gesetzt. Der Bereich der Variablendeklaration wird von VAR und END_VAR umschlossen. Das Beispiel zeigt einige Variablendeklarationen und -definitionen:

```
1   (* einige Variablendeklarationen und -initialisierungen *)
2   VAR
3      a       : INT;
4      b       : DINT := 5;
5      c, d, e : BYTE := 100;
6      Taster  : BOOL := false;
7      Zeit    : TIME;
8      Zaehler : CTUD;
9   END_VAR
```

Beispiel einiger Variablendeklarationen

Generell werden Variablen von Ihrem Typ durch einen Doppelpunkt getrennt. Nach dem Typ kann die Deklaration durch ein Semikolon abgeschlossen werden. In diesem Fall erhält die Variable den Standard-Initialisierungswert, der gleich 0 ist. Das Semikolon darf auch erst nach der Zuweisung (Operator „:=“) eines Initialwertes erfolgen, wie dies in den Zeilen 4 bis 6 geschehen ist (die Variablen in Zeile 5 erhalten dabei alle den Initialwert 100). Funktionsbausteine und Eingangsvariablen (s. u.) können keinen Initialwert

erhalten!

> Variablen werden vom Compiler semantisch von Sprungmarken unterschieden. Sie könnten daher eine Sprungmarke exakt so benennen wie schon einen existierenden Variablenbezeichner. Bitte vermeiden Sie diese Eigenheit, da sie zu unleserlichem Quelltext führt und in folgenden Versionen höchst wahrscheinlich zu einem Fehler.

6.3.1 Definition von konstanten Variablen

Der Begriff der „konstanten Variablen“ ist in sich schon ein Widerspruch - gemeint sind hier aber Bezeichner, die einen nicht veränderbaren Wert besitzen sollen.

Im Unterschied zu den Variablen wird durch den Compiler sichergestellt, dass der Wert eines Konstantenbezeichners nicht verändert werden kann. Der unten stehende Codeausschnitt zeigt die Definition einiger Konstanten:

```
1   (* einige Konstantendefinitionen *)
2   VAR CONSTANT
3      PI_mal_2 : REAL := 6.24;
4      MAX      : BYTE := 100;
5      MIN      : BYTE := 25;
6   END_VAR
```

Beispiel einiger Konstantendefinitionen

Wie aus dem obigen Beispiel zu sehen ist, dient das Schlüsselwort `CONSTANT` als Unterscheidungsmerkmal zur Variablendeklaration. Sie können also in einem durch `VAR` und `END_VAR` eingeschlossenen Abschnitt nur Variablen oder nur Konstanten definieren.
Konstanten ohne Initialwert erzeugen beim Kompilieren eine Warnmeldung!

6.3.2 Eingänge festlegen

Die Eingänge des Soft-SPS-Blockes werden ebenfalls innerhalb des Quelltextes deklariert. Hierzu dient ein durch `VAR_INPUT` und `END_VAR` eingegrenzter Block. Da die Eingänge von der BORIS-Simulationsstruktur gespeist werden, macht es keinen Sinn, Initialwerte für diese Variablen definieren zu können. Der Compiler erzeugt in diesem Fall eine entsprechende Warnung. Des Weiteren können keine Funktionsbausteine innerhalb der Eingangsdeklaration verwendet werden.

BORIS arbeitet immer mit 80 Bit breiten Fließpunktzahlen, die Soft-SPS konvertiert die vom BORIS-System erhaltenen Werte in den angegebenen Datentyp. Dabei werden drei Fälle unterschieden:

Datentyp	**Konvertierter Eingangswert / *E*:**
`BOOL`	TRUE, wenn *E* >2.5 FALSE sonst.
`BYTE, WORD, DWROD, INT, DINT, TIME`	*E* wird zu zur nächstgelegenen Ganzzahl gerundet. Die Ganzzahl wird schließlich der Variable zugewiesen, wobei keinerlei Bereichsüberprüfung erfolgt (falls notwendig, kann diese sehr einfach durch den Begrenzer-Block unter BORIS vorgenommen werden). Bei dem Datentyp `TIME` wird davon ausgegangen, dass *E* eine Anzahl an Millisekunden enthält.
`REAL`	Auflösungsreduktion: Der 80-Bit-Fließkommawert wird in einen 32-Bit-Fließkommawert umgewandelt. Diese Umwandlung ist mit Rundungsfehlern behaftet!

6.3.3 Ausgänge festlegen

Ähnlich wie bei den Eingängen gibt es für die Deklaration der Ausgänge des Soft-SPS-Blockes einen

eigenen Bereich. Dieser wird durch die Schlüsselworte VAR_OUTPUT und END_VAR eingeschlossen. Die Verwendung von Funktionsbausteinen ist hier ebenfalls nicht zulässig. Im Gegensatz zu den Eingängen können aber Initialwerte für die Variablen angegeben werden.

Die Ausgangsvariablen der Soft-SPS werden in die von BORIS benötigten 80 Bit breiten Fließpunktzahlen gewandelt. Dabei erfolgt die Konvertierung folgendermaßen:

Datentyp	**Konvertierung in den Ausgangwert *A***
BOOL	Ist der Wert >2.5, so wird *A=5* *A=0* sonst.
BYTE, WORD, DWORD, INT, DINT, REAL, TIME	Sind direkt als Wert in *A* darstellbar und werden entsprechend übernommen. Für den Typ TIME wird die Anzahl der Millisekunden ausgegeben.

6.4 Eingebaute Variablen

Ihnen stehen einige fest eingebaute Variablen (ohne dass diese deklariert werden müssen) zur Verfügung. Die folgende Tabelle erklärt diese:

Variablenname	**Typ**	**CONSTANT**	**Wert**
_SIMTIME	TIME	Nein	Enthält die aktuelle Simulationszeit unter BORIS. Diese Variable läuft nach einer Simualtionszeit von 4294920 Sekunden (entspricht etwas mehr als 49 Tage) über.
_CYCLE	DWORD	Nein	Enthält die Anzahl der bisherigen Simulationsschritte; nach 4294967296 Schritten fängt dieser Zähler wieder bei 0 an.
_PI	REAL	Ja	die Kreiszahl π (≈3.1415927)
%M*n*.*m* oder auch *%MXn.m*	BOOL	Nein	Referenziert das Bit *m* des Bytes *n* des Merkerspeichers
%MB*n*	BYTE	Nein	Referenziert das Byte *n* des Merkerspeichers
%MW*n*	WORD	Nein	Referenziert 2 Bytes als ein WORD des Merkerspeichers ab dem Byte *n*
%MD*n*	DWORD	Nein	Referenziert 4 Bytes als ein DWORD des Merkerspeichers ab dem Byte *n*

Die Merkerspeicherreferenzen sind nur dann vorhanden, wenn diese im Quelltext verwendet werden. Sie dürfen und können auch nicht deklariert werden. Alle anderen Variablen sind immer verfügbar.

6.5 Merkerspeicher

In einigen Fällen benötigt der Programmierer die Möglichkeit, einzelne Bits einer Variablen auszuwerten. Dieses kann sehr einfach geschehen, wenn die Variable in den Merkerspeicher geschrieben wird. Statt logischer Verknüpfungen und Prüfungen auf 0 kann jedes Bit der Variable direkt angesprochen werden, was die Auswertung sehr vereinfacht. Da der Merkerspeicher in einem zugehörigen Fenster angezeigt werden kann (s. Kapitel Merkerspeicher anzeigen und verändern), ist eine anschauliche Repräsentation im Binärformat möglich.
Der Merkerspeicher wird zu Beginn einer Simulation gelöscht, d. h. alle Bits werden auf Null gesetzt. Es stehen die unter Eingebaute Variablen erläuterten Merkerspeicherreferenzen zur Verfügung.

6.6 Automatische Typkonvertierung

Die automatische Typkonvertierung tritt in Kraft, sobald zwei unterschiedliche Typen miteinander verknüpft werden. Bei den arithmetischen Operationen ADD, SUB, allen Vergleichen (LT, LE, EQ, NE, GE und GT) und Bitverknüpfungsoperationen (AND, OR, XOR, etc.) sowie allen zugehörigen Klammeroperatoren erfolgt die Typkonvertierung nach dem Prinzip des kleinsten gemeinsamen Darstellungsbereiches (INT → DINT; bzw. für vorzeichenlose Zahlen BYTE → WORD → DWORD). Lässt sich dieses Prinzip nicht anwenden, so wird konvertiert wie in der folgende Tabelle dargestellt:

Typ des Akkus	**Typ des Operanden**	**Ergebnistyp**
REAL	Alle außer TIME	REAL
alle außer TIME	REAL	REAL
DINT	Alle außer TIME	DINT
alle außer TIME	DINT	DINT
DOWRD	INT	DINT
INT	DWORD	DINT
INT	WORD	DINT
WORD	INT	DINT
BYTE	INT	INT
INT	BYTE	INT

Additive Operatoren und Vergleiche mit dem Typ Time sind nur zulässig, wenn beide Operanden diesen Typ besitzen. Des Weiteren sind folgende implizite Konvertierungen bemerkenswert:

- Bei der Multiplikation durch MUL bzw. MUL(wird zusätzlich der Ergebnistyp vergrößert (sofern dies möglich ist), damit ein Überlauf möglichst vermieden wird.
- Die Division (DIV, DIV( ) verändert den Typ des Akkumulators nicht, es sei denn, es handelt sich bei Divisor und Dividend um den Typ Time; in diesem Fall ist das Ergebnis vom Typ REAL.
- Die numerischen Operationen SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS und ATAN resultieren alle in den Typ REAL.

Jeder Typ kann in den Typen BOOL gewandelt werden, wenn es notwendig ist. Diese Eigenschaft erlaubt es, die bedingten Befehle wie JMPC, RETC, STC, CALC und deren Negationen mit jedem Typen zu verwenden ohne zuvor eine explizite Typumwandlung vornehmen zu müssen. Dabei gilt: Ist der Wert einer Variablen von 0 verschieden, so evaluiert die Konvertierung dieses Wertes zu true, sonst false.

> Wenn Sie den Typen REAL nach einer Rechenoperation für eine Auswertung verwenden (also implizit oder explizit zum Typ BOOL konvertieren), sollten Sie diesen besser auf einen Wertebereich um 0 prüfen, um Rundungsfehler zu kompensieren!

6.7 Anweisungen

In AWL darf jede Zeile maximal eine Anweisung enthalten. Eine Anweisung besteht dabei nach IEC 61131-3 aus einer optionalen Sprungmarke, der Anweisung, dem Operanden und einem optionalen Kommentar. Die Soft-SPS ist nicht ganz so streng, sie erlaubt es, dass die Sprungmarke nicht in der selben Zeile stehen muss und lässt es zu, einen Kommentar beliebig zu platzieren. Dabei ist eine Sprungmarke ein Bezeichner, dem unmittelbar ein Doppelpunkt folgt und ein Kommentar eine beliebige durch (* und *) eingeschlossene Zeichenkette, die sich ggf. auch über mehrere Zeilen erstreckt. Als Operanden dürfen je nach Anweisung Sprungziele, lexikalische Konstanten bzw. Variablen und/oder Konstanten stehen. Nahezu alle Anweisungen

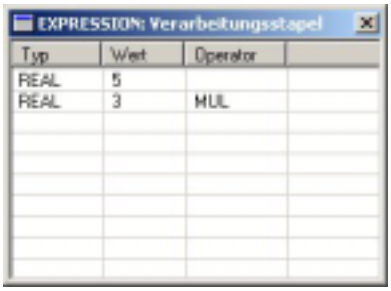
arbeiten mit einem so genannten Verabeitungsstapel.

6.7.1 Verarbeitungsstapel

Die Soft-SPS arbeitet nach dem Prinzip eines Verarbeitungsstapels. Anweisungen operieren immer mit diesem Stapel, genauer gesagt, mit dem obersten Element des Stapels. Das oberste Element des Stapels wird allgemein *aktuelles Ergebnis* (kurz: *AE*) genannt.

Meistens besteht der Stapellediglich aus einem Operanden; erst wenn Anweisungen mit einer Klammer verwendet werden, wird daraus ein *richtiger* Stapel. Dieser beinhaltet dann auch zum Wert eines Operanden den Operator, mit dem eine spätere Verknüpfung durchgeführt wird.

Der Verarbeitungsstapel kann während des Debuggings angezeigt werden. In der Anzeige wächst der Stapel nach unten!



Anzeigefenster des Verarbeitungsstapels

Die angezeigten Werte können, sofern sie nicht vom Typ `BOOL`, `REAL` oder `TIME` sind, unterschiedlich dargestellt werden. Hierzu betätigen Sie innerhalb des Fensters die rechte Maustaste und wählen dann aus dem erscheinenden Popupmenü aus, ob die Werte binär, oktal, dezimal oder hexadezimal dargestellt werden sollen (vgl. Variablen beobachten/verändern).

Wie oben schon erwähnt bildet sich erst ein richtiger Stapel, wenn Anweisungen ausgeführt werden die eine öffnende Klammer besitzen. Diese Klammer wird auch als *Modifizierer* der Anweisung bezeichnet.

6.7.2 Modifizierer einer Anweisung

Es gibt drei sogenannte Modifizierer, die direkt hinter **einigen** Anweisungen auftreten dürfen. Das Wissen um deren Bedeutung erhöht die Lesbarkeit einer AWL.

N Negiert den Operanden.

C Die Ausführung dieser Anweisung erfolgt nur, wenn das *AE* zu `TRUE` evaluiert.

(Legt den Operanden und die Verknüpfungsoperation zwischen Operand und *AE* zur späteren Ausführung auf den Stapel.

Zur Erläuterung erfolgt ein Beispiel:

```
1   (*   Einfache Berechung mit Klammern:
2        o1:= i1*(i2-i3) , falls NotOk = false ist
3        sonst bleibt o1 unverändert.
4   *)
5   VAR_INPUT
6      Notok : BOOL;
7      i1 : INT;
8      i2 : INT;
9      i3 : INT;
10  END_VAR
11
12  VAR_OUTPUT
13     o1 : REAL;
14  END_VAR
15
16          LDN     Notok
17          JMPC    END
18          LD      i1
19          MUL(    i2
20          SUB     i3
21          )
22          ST      o1
23  END:    RET
```

Beispiel für Anweisungen mit Modifizierern

Die Anweisung `LDN` (Anweisung `LD` + Modifizierer `N`) in Zeile 16 lädt den negierten Wert der Variable `NotoK`. Eine Zeile weiter steht die bedingte Ausführung eines Sprunges durch `JMPC`. Die Zeile 19 enthält eine Multipikation mit dem Modifizierer `(`, diese wird erst mit der schließenden Klammer in Zeile 21 ausgeführt.

6.7.3 AWL Anweisungen

Hier soll ein Überblick über die Anweisungen und deren Wirkung gegeben werden. Eine detaillierte Beschreibung finden Sie im Kapitel AWL-Befehle. Zur besseren Übersicht und kürzeren Schreibweise werden hier zunächst Mengen von Datentypen definiert:

ANY	meint alle Datentypen
NONE	meint keinen Datentyp bzw. keinen Operanden
ANY_NUM	meint alle numerischen Typen, also (`BYTE`, `WORD`, `DWORD`, `INT`, `DINT`, `REAL`)
ANY_SIGNED	meint alle vorzeichenbehafteten Typen, also (`INT`, `DINT`, `REAL`)
ANY_INT	meint alle ganzzahligen Typen, also (`BYTE`, `WORD`, `DWORD`, `INT`, `DINT`)
ANY_BIT	meint alle Typen, die ein binäre Darstellung besitzen, also (`BIT`, `BYTE`, `WORD`, `DWORD`, `INT`, `DINT`)
Label	meint ein Sprungziel
FB	meint eine Variable, die als Funktionsbaustein deklariert ist

Laden und Speichern

Anweisung	Datentypen	Kurzbeschreibung
`LD`	*ANY*	Lädt den Wert des Operanden in das *AE*. Der alte Wert des *AE* wird dabei überschrieben.
`LDN`	*ANY_BIT*	Wie `LD`, nur mit negiertem Operanden.
`ST`	*ANY*	Speichert den Wert des *AE* in den Operanden.
`STN`	*ANY_BIT*	Speichert den negierten Wert des *AE* in den Operanden.
`STC`	*ANY*	Speichert `TRUE` in den Operanden, falls das *AE* zu `TRUE` ausgewertet werden kann.

STCN	*ANY*	Speichert FALSE in den Operanden, falls das *AE* zu TRUE ausgewertet werden kann.

Sprünge

Anweisung	Datentypen	Kurzbeschreibung
JMP	*LABEL*	Springt unbedingt zum angegebenen Sprungziel.
JMPL	*NONE*	Springt so viele Anweisungen nach vorn, wie durch den Wert des *AE* angegeben wird. Mindestens eine JMP-Anweisung muss dieser Anweisung folgen (das L des Befehls steht für List, gemeint ist eine Sprungliste, ähnlich dem *switch* aus *C* bzw. dem *case* aus *Pascal*).
JMPC	*LABEL*	Springt zum angegebenen Sprungziel, wenn das *AE* zu TRUE ausgewertet werden kann.
JMPCN	*LABEL*	Springt zum angegebenen Sprungziel, wenn das *AE* zu FALSE ausgewertet werden kann.
RET	*NONE*	Beendet die Ausführung der AWL.
RETC	*NONE*	Beendet die Ausführung der AWL, wenn das *AE* zu TRUE ausgewertet werden kann.
RETCN	*NONE*	Beendet die Ausführung der AWL, wenn das *AE* zu FALSE ausgewertet werden kann.

Funktionsbausteinaufrufe

Anweisung	Datentypen	Kurzbeschreibung
CAL	*FB*	Führt den angegebenen Funktionsbaustein aus.
CALC	*FB*	Führt den angegebenen Funktionsbaustein aus, wenn das *AE* zu TRUE evaluiert.
CALCN	*FB*	Führt den angegebenen Funktionsbaustein aus, wenn das *AE* zu FALSE evaluiert.

Bitverknüpfungen

Anweisung	Datentypen	Kurzbeschreibung
AND, OR, XOR	*ANY_BIT*	Führt die entsprechende Bitverknüpfung zwischen dem *AE* und dem Operanden aus.
ANDN, ORN, XORN	*ANY_BIT*	Führt die entsprechende Bitverknüpfung zwischen dem *AE* und dem negierten Operanden aus.
AND(, OR(, XOR(,ANDN(, ORN(, XORN(	*ANY_BIT*	Legt Operand und Operation auf den Verarbeitungsstapel.
NOT	*NONE*	Kehrt die Bits des *AE* um. Die, die vorher 1 waren, sind danach 0 und umgekehrt.

Schiebeoperationen

Anweisung	Datentypen	Kurzbeschreibung
SHL, SHR	BYTE, WORD, DWORD	Verschiebt das Bitmuster des *AE* (muss vom Typ *ANY_INT* sein) um so viele Bits nach links bzw. rechts, wie durch den Wert des

		Operanden angegeben wird.
ROL, ROR	BYTE, WORD, DWORD	Rotiert das Bitmuster des *AE* (muss vom Typ *ANY_INT* sein) um so viele Bits links bzw. rechts herum, wie durch den Wert des Operanden angegeben wird.

Vergleiche

Anweisung	Datentypen	Kurzbeschreibung
LT, LE, EQ, NE, GE, GT	*ANY*	Prüfung, ob das *AE* kleiner, kleiner oder gleich, gleich, ungleich, größer oder gleich bzw. größer als der Wert des Operanden ist. Im AE steht anschließend der Datentyp BOOL.
LT(, LE(, EQ(, NE(, GE(, GT(	*ANY*	Legen Operand und Operation auf den Verarbeitungsstapel.

Schließung einer Klammerebene

Anweisung	Datentypen	Kurzbeschreibung
)	*NONE*	Führt die oberste auf dem Verarbeitungsstapel liegende Operation mit den obersten beiden Operanden durch.

Arithmetische Funktionen

Anweisung	Datentypen	Kurzbeschreibung
ADD, SUB, MUL, DIV, MOD	*ANY_NUM*	Führt eine Additions-, Multiplikations-, Subtraktions-, Divisions- oder Modulo-Operation (Ermittlung des Restes einer Division) aus.
ADD(, SUB(, MUL(, DIV(, MOD(	*ANY_NUM*	Legt Operand und Operation auf den Verarbeitungsstapel.
ADD, SUB	TIME	Addiert/Subtrahiert zwei Zeiten. *AE* und Operand müssen den Typ TIME besitzen.
ADD(, SUB(	TIME	Legt Operand und Operation auf den Verarbeitungsstapel. Bei der korrespondierenden schließenden Klammer müssen zwei Zeitwerte als oberstes auf dem Verarbeitungsstapel liegen.
MUL	TIME	Multipliziert eine Zeitangabe mit einem numerischen Wert (*ANY_NUM*).
MUL(	TIME	Legt Operand und Operation auf den Verarbeitungsstapel. Bei der korrespondierenden schließenden Klammer müssen ein Zeitwert und ein *ANY_NUM* als oberstes auf dem Verarbeitungsstapel liegen.
DIV	TIME	Zeitdivision; folgende Berechnungen sind zulässig TIME / *ANY_NUM* → TIME TIME / TIME → REAL
DIV(	TIME	Legt Operand und Operation auf den Verarbeitungsstapel. Bei der korrespondierenden schließenden Klammer müssen die zu verknüpfenden Datentypen denen unter DIV entsprechen.
MOD	TIME	Ermittelt die Restzeit der Division zweier Zeitangaben. *AE* und Operand müssen den Typ TIME besitzen.

MOD(	TIME	Legt Operand und Operation auf den Verarbeitungsstapel. Bei der korrespondierenden schließenden Klammer müssen die zu verknüpfenden Datentypen denen unter MOD entsprechen.

Numerische Funktionen

Die numerischen Funktionen arbeiten alle mit dem aktuellen Ergebnis *AE*. Da diese Operationen keinen Operanden besitzen, bezieht sich die Spalte des Datentyps auf das *AE*.

Anweisung	**Datentypen (*AE*)**	**Kurzbeschreibung**
ABS	*ANY_NUM*	Bildet den Absolutbetrag des *AE* (der Typ vom *AE* bleibt unverändert).
NEG	*ANY_NUM*	Negiert das *AE* (*AE* wird zu INT, DINT oder REAL).
LN	*ANY_NUM*	natürlicher Logarithmus (*AE* wird zu REAL)
LOG	*ANY_NUM*	Logarithmus zur Basis 10 (*AE* wird zu REAL)
EXP	*ANY_NUM*	Exponentialfunktion (*AE* wird zu REAL)
SQRT	*ANY_NUM*	Quadratwurzel (*AE* wird zu REAL)
SIN	*ANY_NUM*	Sinusfunktion (*AE* wird zu REAL)
COS	*ANY_NUM*	Kosinusfunktion (*AE* wird zu REAL)
TAN	*ANY_NUM*	Tangensfunktion (*AE* wird zu REAL)
ASIN	*ANY_NUM*	Umkehrfunktion zu Sinus (*AE* wird zu REAL)
ACOS	*ANY_NUM*	Umkehrfunktion zu Kosinus (*AE* wird zu REAL)
ATAN	*ANY_NUM*	Umkehrfunktion zu Tangens (*AE* wird zu REAL)

Typumwandlung

Da die Soft-SPS über eine mächtige implizite Typkonvertierung verfügt, sind die hier aufgeführten Befehle selten notwendig. Bei der expliziten Typkonvertierung erfolgt generell keine Überlaufprüfung (s. Kapitel Prüfungen).

Anweisung	**Möglichkeiten für *xxxx***
BOOL_TO_*xxxx*	BYTE, WORD, DWORD, INT, DINT, REAL
BYTE_TO_*xxxx*	BOOL, WORD, DWORD, INT, DINT, REAL, TIME
WORD_TO_*xxxx*	BOOL, BYTE, DWORD, INT, DINT, REAL, TIME
DWORD_TO_*xxxx*	BOOL, BYTE, WORD, INT, DINT, REAL, TIME
INT_TO_*xxxx*	BOOL, BYTE, WORD, DWORD, DINT, REAL, TIME
DINT_TO_*xxxx*	BOOL, BYTE, WORD, DWORD, INT, REAL, TIME
REAL_TO_*xxxx*	BOOL, BYTE, WORD, DWORD, INT, DINT, TIME
TIME_TO_*xxxx*	BOOL, BYTE, WORD, DWORD, INT, DINT, REAL

6.8 Standard-Funktionsbausteine

Die Soft-SPS verfügt über drei verschiedenen Funktionsbausteintypen. Diese lassen sich in weitere spezielle Funktionalitäten unterteilen. Die drei Bausteintypen sind:

- Flankendetektoren,
- Zähler und
- Timer,

die im Folgenden näher beschrieben werden. Zur Verwendung von Funktionsbausteinen muss wie folgt vorgegangen werden:

1. Deklaration einer Variablen mit dem Typ des gewünschten Funktionsbausteins (Erzeugung einer Instanz eines Funktionsbausteins)
2. Definition der Eingänge der Funktionsbausteininstanz
3. Aufruf durch CAL bzw. CALC oder CALCN
4. Auswertung des Ergebnisses durch das Auslesen der Ausgänge der Bausteininstanz

Die Ein- und Ausgänge eines Funktionsbausteines werden über seine Instanzvariable (Punkt 1) angesprochen. Dabei wird dem Variablenbezeichner ein Punkt angehängt, dem unmittelbar der Bezeichner der entsprechenden Bausteinvariablen folgt (s. u.).

6.8.1 Flankendetektoren

Häufig muss auf eine Flanke an einem Eingang reagiert werden. In diesen und ähnlichen Fällen sollten Sie von den hier beschriebenen Detektoren Gebrauch machen. Da es eine steigende und eine fallende Flanke gibt, gibt es genau zwei Flankendetektoren: R_TRIG und F_TRIG.

R_TRIG

Dient der Erkennung von steigenden Flanken. Der Funktionsbaustein liefert am Ausgang HIGH, wenn am Eingang eine steigende Flanke erkannt worden ist.

Eingangsvariablen	Datentyp	Beschreibung
CLK	BOOL	Ist mit dem Wert der Variablen zu beschreiben, die auf eine steigende Flanke untersucht werden soll.

Ausgangsvariablen	Datentyp	Beschreibung
Q	BOOL	Enthält TRUE, wenn eine steigende Flanke erkannt wurde.

Es wird schon beim ersten Aufruf eine Flanke erkannt, sofern CLK den Wert TRUE besitzt.

F_TRIG

Der Funktionsbaustein F_TRIG dient der Erkennung von fallenden Flanken. Er liefert am Ausgang HIGH, wenn am Eingang eine fallende Flanke erkannt worden ist.

Eingangsvariablen	Datentyp	Beschreibung
CLK	BOOL	Ist mit dem Wert der Variablen zu beschreiben, die auf eine fallende Flanke untersucht werden soll

Ausgängsvariablen	Datentyp	Beschreibung
Q	BOOL	Enthält TRUE, wenn eine fallende Flanke erkannt wurde.

Es wird schon beim ersten Aufruf eine Flanke erkannt, sofern CLK den Wert FALSE besitzt.

Beispiel

Die Verwendung der beiden Flankendetektoren zeigt der folgende Quelltext, der auch als Beispiel *Trigger.AWL* vorhanden ist:

```
1   (*
2      Nachbau des "Logischen Flankendetektors" von BORIS
3      mit zusätzlichem Eingang (Mode), der die Betriebsart
4      festlegt:
5
6      Mode=0: Ausgang TRUE, wenn steigende Flanke
7      Mode=1: Ausgang TRUE, wenn fallende Flanke
8      Sonst : Ausgang TRUE, wenn steigende oder fallende Flanke
9   *)
10  VAR_INPUT
11     Mode   : BYTE;    (* Betriebsart *)
12     INPUT  : BOOL;    (* auf Flanken zu untersuchender Eingang *)
13  END_VAR
14
15
16  VAR_OUTPUT
17     OUTPUT : BOOL;
18  END_VAR
19
20  VAR
21    SF      : R_TRIG; (*SF = steigende Flanke *)
22    FF      : F_TRIG; (*FF = fallende Flanke  *)
23  END_VAR
24
25
26             (* Initialisierungsteil der AWL: *)
27              LD       INPUT
28              ST       SF.CLK
29              ST       FF.CLK
30              LD       Mode
31              JMPL
32              JMP      STEIGEND
33              JMP      FALLEND
34              JMP      BEIDES
35
36  STEIGEND:   CAL      SF
37              LD       SF.Q
38              ST       OUTPUT
39              RET
40
41  FALLEND:    CAL      FF
42              LD       FF.Q
43              ST       OUTPUT
44              RET
45
46  BEIDES:     CAL      SF
47              CAL      FF
48              LD       SF.Q
49              OR       FF.Q
50              ST       OUTPUT
51              RET
```

Beispiel zur Verwendung der Flankendetektoren

Die Deklaration der beiden Flankendetektoren erfolgt in den Zeilen 21 und 22. Es werden zwei Variablen, `SF` und `FF`, als Instanzen der Funktionsbausteine, `R_TRIG` und `F_TRIG`, erzeugt. In den Zeilen 28 und 29 wird den Eingängen `CLK` der Funktionsbausteininstanzen der Wert der Eingangsvariablen `INPUT` zugewiesen. Der eigentliche Aufruf der Funktionsbausteine erfolgt, abhängig vom Wert der Eingangsvariablen `MODE`, in den Zeilen 36 und 41 für eine der beiden und in den Zeilen 46 und 47 für beide Instanzen. Jeweils im Anschluss dieser Zeilen befinden sich Anweisungen zum Auslesen des/der Ergebnisse(s) von `SF` und/oder `FF`, das sich jeweils in der Bausteinvariable `Q` befindet.

6.8.2 Zähler

Zähler sind ein wichtiges Hilfsmittel für eine SPS. Immer wenn es darum geht, flankengesteuert einen Wert zu erhöhen oder zu verringern, sollten Sie einen Zähler einsetzen. Sie haben bei Zählern die Möglichkeit, den Zählbereich einzuschränken und ihn zurückzusetzen. Es gibt drei Varianten an Zählern: einen Vorwärtszähler `CTU`, einen Rückwärtszähler `CTD` und einen, der beide Richtungen unterstützt, `CTUD`.

CTU

Der Funktionsbaustein CTU ist ein reiner Vorwärtszähler, der bei einer steigenden Flanke solange zählt, bis sein Maximum erreicht ist. Der Zähler beginnt immer bei 0 und lässt sich zurücksetzen.

Eingangsvariablen	Datentyp	Beschreibung
CU	BOOL	flankengesteuerter Zähleingang
R	BOOL	Ist dieser Eingang TRUE, so werden CV = 0 und QU = FALSE gesetzt. Der Zähler ignoriert außerdem alle eingehenden Flanken, bis FALSE anliegt.
PV	INT	Maximaler Wert des Zählers.

Ausgangsvariablen	Datentyp	Beschreibung
QU	BOOL	Enthält TRUE, wenn der Maximalwert des Zähler (PV) erreicht wurde.
CV	INT	Enthält den aktuellen Zählerstand.

CTD

Das Gegenstück zum Vorwärtszähler ist CTD, ein reiner Rückwärtszähler, der von einem vorgegebenen Wert bis 0 herunter zählt. Es werden steigende Flanken am Eingang gezählt und der Zähler kann mit seinem maximalen Wert geladen werden.

Eingangsvariablen	Datentyp	Beschreibung
CD	BOOL	flankengesteuerter Zähleingang
LD	BOOL	Ist dieser Eingang TRUE, so werden CV = PV und QD = FALSE gesetzt. Der Zähler ignoriert außerdem alle eingehenden Flanken, bis FALSE anliegt.
PV	INT	Maximaler Wert des Zählers.

Ausgangsvariablen	Datentyp	Beschreibung
QD	BOOL	Enthält TRUE, wenn der Zähler 0 erreicht hat.
CV	INT	Enthält den aktuellen Zählerstand.

CTUD

Dieser Funktionbaustein stellt eine Kombination aus den beiden Zählern CTU und CTD zur Verfügung. Er wird eingesetzt, wenn es erforderlich ist, beide Zählrichtungen zu berücksichtigen. Der Baustein hat je einen flankengesteuerten Eingang zum Vorwärts- und Rückwärtszählen. Zusätzlich kann er sowohl auf 0 als auch auf den Maximalwert gesetzt werden.

Eingangsvariablen	Datentyp	Beschreibung
CU	BOOL	flankengesteuerter Vorwärtszähleingang

CD	BOOL	flankengesteuerter Rückwärtszähleingang
R	BOOL	Liegt an diesem Eingang TRUE, so werden CV = 0, QU = FALSE und QD = TRUE gesetzt. Der Zähler ignoriert außerdem alle eingehenden Flanken, bis FALSE anliegt.
LD	BOOL	Liegt an diesem Eingang TRUE, so werden CV = PV, QU = TRUE und QD = FALSE gesetzt. Der Zähler ignoriert außerdem alle eingehenden Flanken, bis FALSE anliegt.
PV	INT	Maximaler Wert des Zählers.

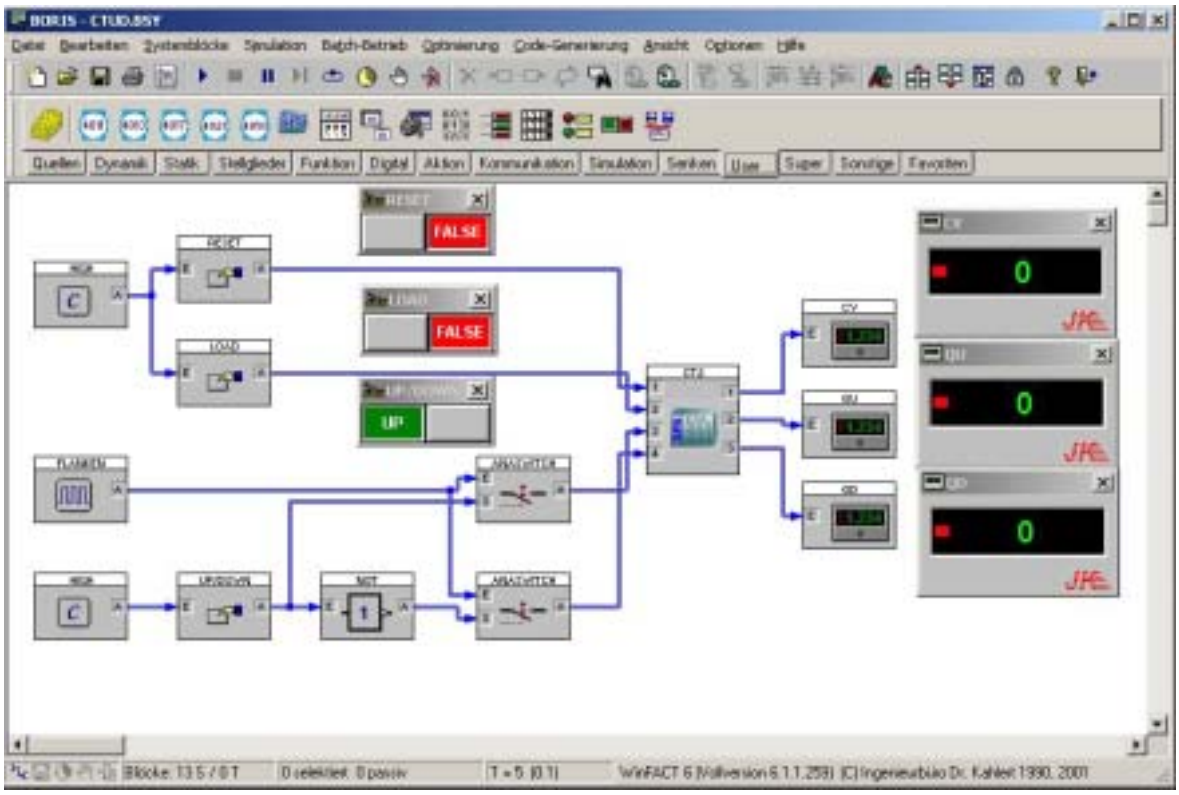
Ausgangsvariablen	**Datentyp**	**Beschreibung**
QU	BOOL	Enthält TRUE, wenn der Zähler den Wert von PV erreicht hat.
QD	BOOL	Enthält TRUE, wenn der Zähler 0 erreicht hat.
CV	INT	Enthält den aktuellen Zählerstand.

Zwei Dinge gibt es noch bei der Verwendung von CTUD zu beachten:

1. Der Eingang R hat Vorrang vor dem Eingang LD.
2. Wenn sowohl an CU als auch an CD eine Flanke auftritt, bleibt der Zählerstand unbeeinflusst (intern wird er um eins erhöht und sofort wieder um eins erniedrigt).

Beispiel

Das mitgelieferte Beispiel *CTUD.BSY* zeigt die Verwendung des Vorwärts-/Rückwärtszählers. In diesem Beispiel wurden die Ein- und Ausgänge des CTUD über Variablen zu Blockein-/ausgängen gemacht. Sie können mit Hilfe von Druckschaltern Flanken an die Zähleingänge geben und die Wirkung der einzelnen LD- und R-Eingänge nachvollziehen. Die untenstehende Abbildung zeigt die Simulationsstruktur:



Struktur zum Testen des CTUD-Funktionsbausteins (Beispieldatei: CTUD.BSY)

6.8.3 Zeitgeber

In der Soft-SPS sind drei Zeitgeberbausteine realisiert: der Zeitimpuls TP, die Einschaltverzögerung TON und die Ausschaltverzögerung TOF. Sie besitzen alle die gleichen Ein- und Ausgangsvariablen, daher werden diese nur einmalig vorgestellt. Die untenstehenden Diagramme erläutern das Zeitverhalten der einzelnen Zeitgeber.

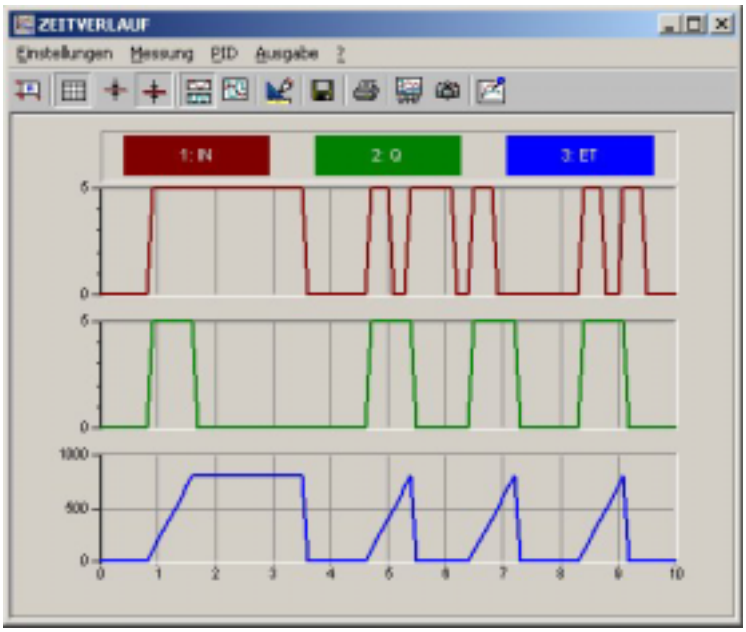
Eingangsvariablen	Datentyp	Beschreibung
IN	BOOL	Eingang, aus dem der Impuls erzeugt bzw. der verzögert werden soll.
PT	TIME	Zeitdauer, die verstreichen soll/muss, um einen Effekt am Ausgang Q zu erzeugen.

Ausgangsvariablen	Datentyp	Beschreibung
Q	BOOL	Ausgang, der den Impuls oder die Verzögerung wiedergibt.
EV	TIME	Enthält die bis jetzt verstrichene Zeit. Kann maximal so groß werden wie PT.

Ist die durch PT angegebene Dauer kleiner als die Abtastschrittweite unter BORIS, so erzeugen die Zeitgeber-Funktionsbausteine einen Fehler. Ist die durch PT angegebene Dauer nicht ganzzahlig durch die Abtastschrittweite teilbar, so wird eine Meldung ausgegeben (s. Kapitel Prüfungen).

TP

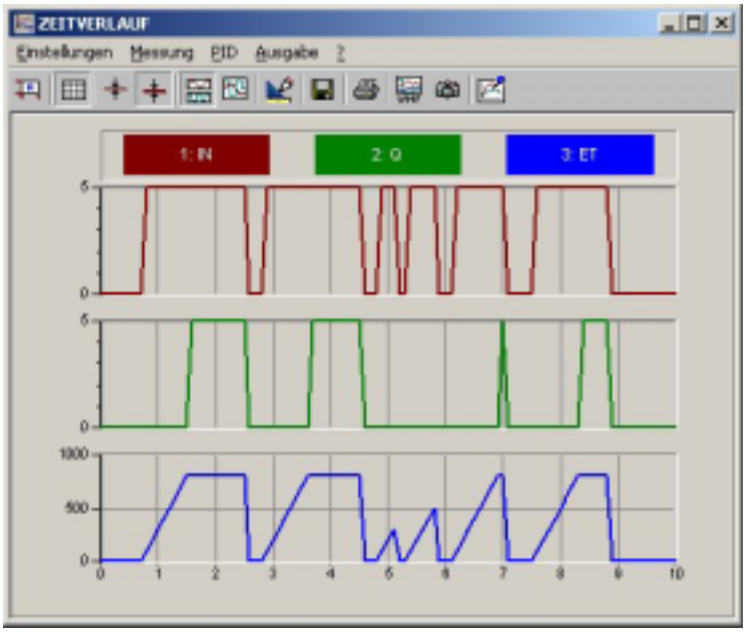
Mit diesem Funktionsbaustein wird ein gleich bleibend langer Impuls am Ausgang Q erzeugt, der bei einer steigenden Flanke am Eingang IN beginnt. Der Impuls kann **nicht** durch eine weitere Flanke am Eingang verlängert werden.



Zeitverhalten für TP abhängig von IN

TON

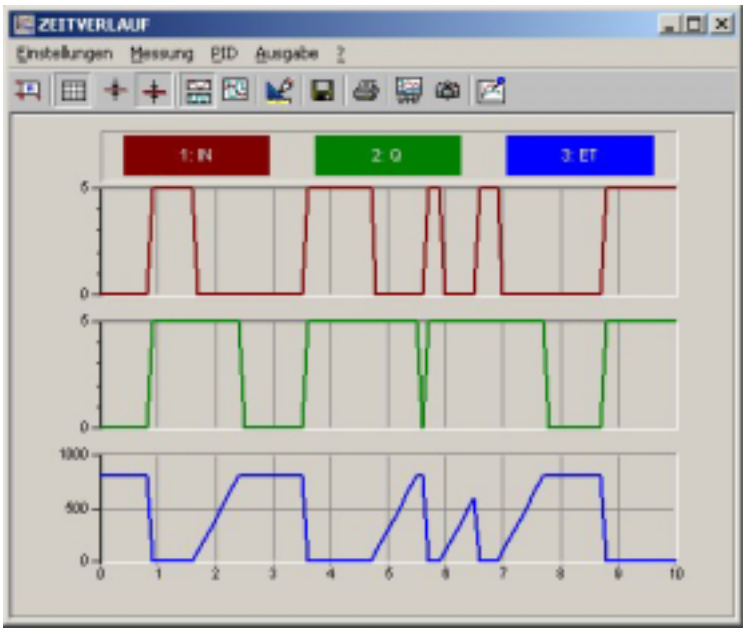
TON stellt eine Einschaltverzögerung dar. Der Eingang IN muss mindestens solange TRUE sein, wie dies in der Eingangsvariablen PT eingestellt ist, um am Ausgang Q den Wert TRUE zu liefern. Sobald der Eingang FALSE wird, wird auch der Ausgang auf FALSE gesetzt.



Zeitverhalten für TON abhängig von IN

TOF

Eine Ausschaltverzögerung lässt sich mit TOF realisieren. Dieser Baustein liefert TRUE an seinem Ausgang Q, sobald der Eingang IN den Wert TRUE hat. Er wird erst dann FALSE, wenn der Eingang FALSE ist **und** die in PT eingestellte Zeit verstrichen ist.



Zeitverhalten für `TOF` abhängig von `IN`

Beispiel

Das für die Zeitgeberbausteine verfügbare Beispiel *TIMER.BSY* wurde verwendet, um die obigen Diagramme zu erstellen. Hier der Quelltext der AWL (*TIMER.AWL*):

```
1.  (*
2.     Beispiel zur Verwendung der Zeitgeberbausteine.
3.     Die unterschiedlichen Zeitgeberbausteine besitzen alle die gleichen Ein- und
4.     Ausgangvariablen, die an den Blockein-/ausgängen abgebildet worden sind.
5.     Daher können Sie durch Verändern der Variablen VZ,
6.     z. B. von TON nach TOF, aus der Einschaltverzögerung eine Auschaltverzögerung
7.     machen.
8.  *)
9.
10. VAR_INPUT
11.   Start : Bool;
12.   Dauer : Word;
13. END_VAR
14.
15.
16. VAR_OUTPUT
17.   Q  : Bool;
18.   ET : TIME;
19. END_VAR
20.
21. VAR
22.   VZ : TON;
23. END_VAR
24.             LD   Start
25.             ST   VZ.IN
26.             LD   Dauer
27.             ST   VZ.PT
28.             cal  VZ
29.             LD   VZ.Q
30.             ST   Q
31.             LD   VZ.ET
32.             ST   ET
```

Beispiel für die Verwendung der Zeitgeberbausteine

7 Einstellungen der AWL zur Ausführung

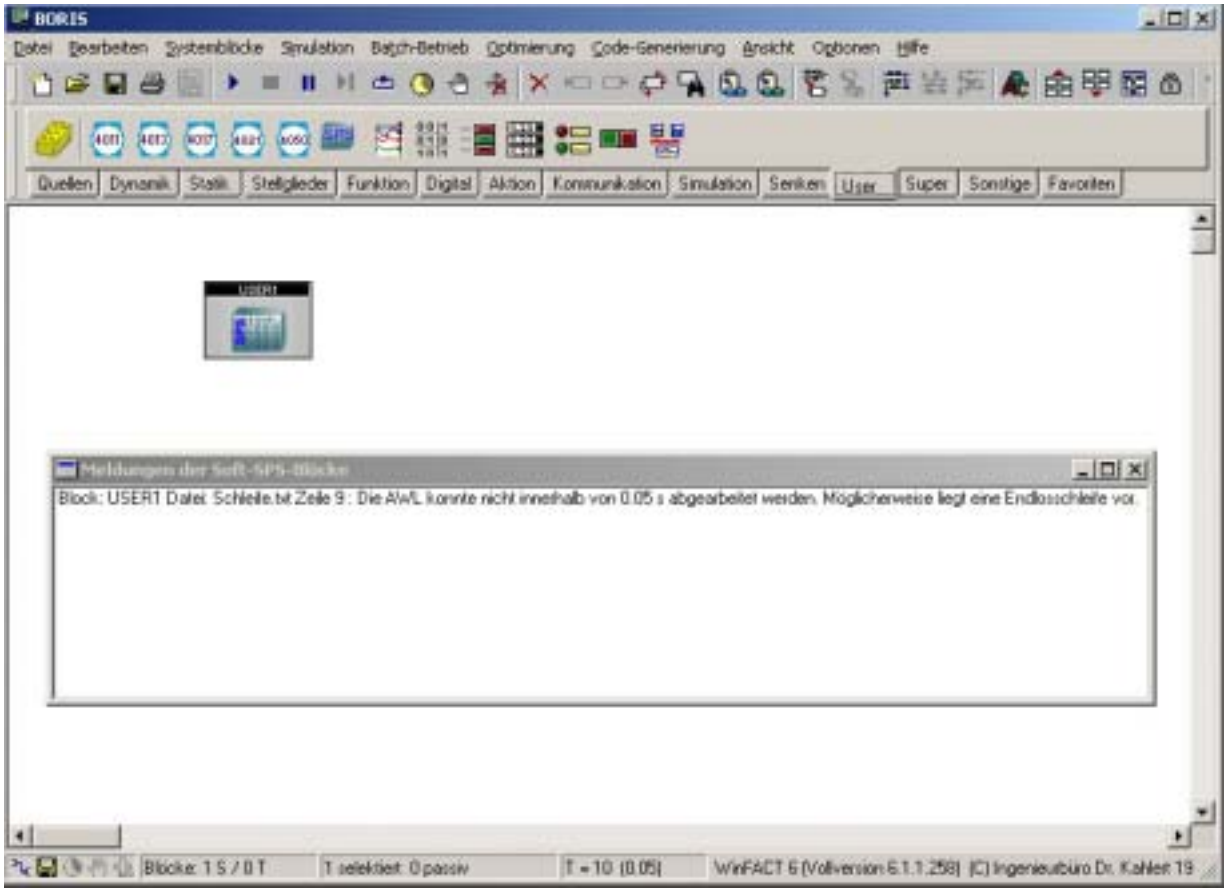
Die syntaktischen und semantischen Fehler werden durch Fehlermeldungen des Compilers angezeigt. Endlosschleifen und Zahlenüberläufe hingegen, können erst zur Laufzeit erkannt werden.

Diese Fehlermeldungen werden in ein globales Meldungsfenster eingetragen, sofern die Entwicklungsumgebung des betreffenden Soft-SPS-Blockes nicht sichtbar (auch nicht als Icon) ist. Das globale Meldungsfenster gibt dabei neben der Fehlermeldung selbst den Blocknamen des Soft-SPS-Blockes, den Dateinamen der AWL und die Zeile, in der der Fehler auftrat, an.

7.1 Zykluszeitüberwachung

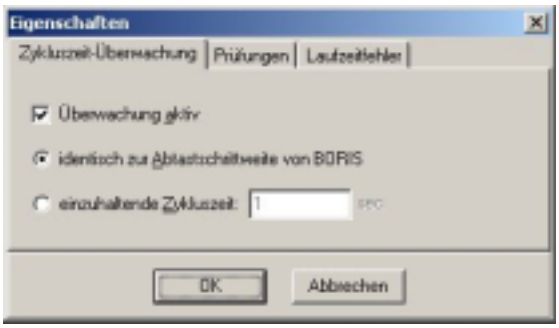
Bei einer Endlosschleife würde der entsprechende Simulationsschritt unter BORIS nicht enden, was bedeutet, dass die Anwendung nicht mehr auf Benutzereingaben reagiert.

Diese Situation kann durch eine Zykluszeitüberwachung vermieden werden. Hierbei wird gefordert, dass die AWL der Soft-SPS in einer bestimmten Zeit abgearbeitet ist. Sollte die Zeit verstrichen sein, die AWL aber noch nicht abgearbeitet, so wird ein Laufzeitfehler mit einer entsprechenden Fehlermeldung generiert. Je nach Einstellung des Verhaltens bei Laufzeitfehlern (s. Kapitel *Verhalten bei Laufzeitfehlern*) erscheint die Entwicklungsumgebung oder das globales Meldungsfenster.



Fehlermeldung über eine nicht eingehaltene Zykluszeit

Die Einstellungen zur Zykluszeitüberwachung können unter START|EIGENSCHAFTEN.. vorgenommen werden und sind durch den folgenden Dialog einstellbar.



Einstellungen zur Zykluszeitüberwachung

In dem Dialog können Sie die einzuhaltende Zykluszeit gleich der Abtastschrittweite von BORIS oder auf einen frei definierbaren Zeitwert setzen. Ein neu eingefügter SPS-Block hat voreingestellt eine aktive Überwachung mit einer zur Abtastzeit identischen Zykluszeit.

Da die Zeitmessung ebenfalls eine gewisse Zeit in Anspruch nimmt, beansprucht die Soft-SPS bei aktiver Zykluszeitüberwachung geringfügig mehr Zeit.

7.2 Prüfungen

Die Soft-SPS unterstützt zwei Prüfungen: die Überlaufprüfung und eine Überprüfung, ob die Zeitdauer von Zeitgeberbausteinen (Bausteinvariable `PT`) mit der eingestellten Abtastschrittweite von BORIS eingehalten werden kann.

Beide sollen nachfolgend besprochen werden.

7.2.1 Überlaufprüfung

Der folgende Quelltext kann als fehlerhaft interpretiert werden, wenn definiert sein soll, dass der Bereich eines ganzzahligen Datentyps nicht überschritten werden darf. Diese Definition ist gegeben, wenn die ÜBERLAUFPRÜFUNG AKTIV ist (Menüpunkt: START|EINSTELLUNGEN... Registerkarte PRÜFUNGEN; siehe Seite *35*).

```
1   VAR
2      I1    : BYTE:=255;
3   END_VAR
4
5
6           LD     I1
7           ADD    1
8           ST     I1
9           RET
```

Quelltext mit einer offensichtlichen Bereichsüberschreitung

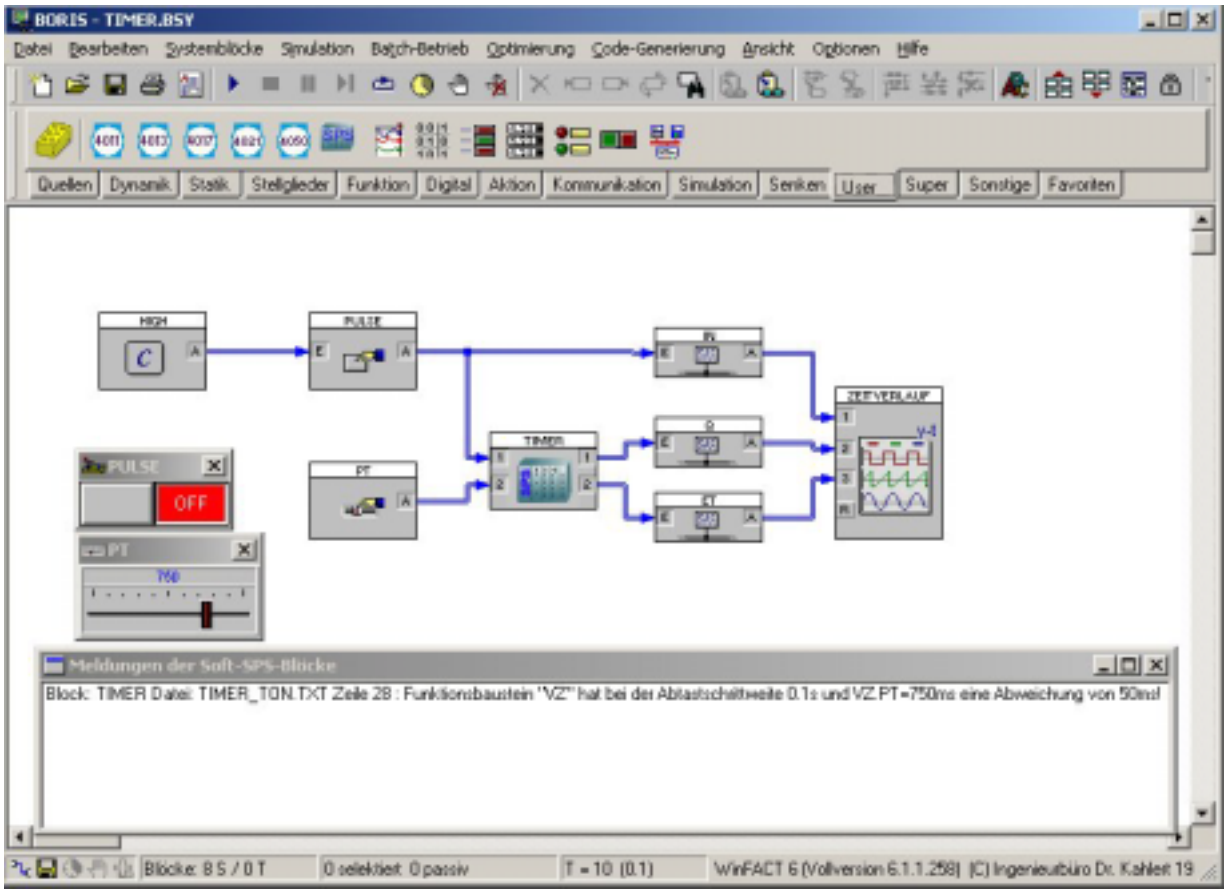
Der Fehler tritt allerdings erst in Zeile 8 und nicht in Zeile 7 auf, da die lexikalische Konstante `1` vom Typ `INT` ist und so das Ergebnis der Addition ebenfalls vom Typ `INT`. Erst bei der impliziten Typumwandlung des `INT` zu `BYTE` fliegt der Fehler auf (die Zahl 256 passt nicht in 8 Bit). Fügen wir zwischen den Zeilen 7 und 8 eine die Anweisung `INT_TO_BYTE` ein, also die entsprechende explizite Typumwandlung, so wird die Bereichsüberprüfung an dieser Stelle nicht vorgenommen. Der Laufzeitfehler ist somit eliminiert, jedoch wird als Ergebnis der Berechnung `0` in `I1` gespeichert.

Die Eingänge der Soft-SPS (s. auch Kapitel Eingänge festlegen) unterliegen nicht dieser Überprüfung. Dies bedeutet, dass die Zuweisung des Wertes `500` an eine Eingangsvariable des Typs `BYTE` in den Wert `500` mod `256` (also `244`) resultiert.

7.2.2 Warnung bei ungenauem Zeitverhalten von Zeitgeberbausteinen

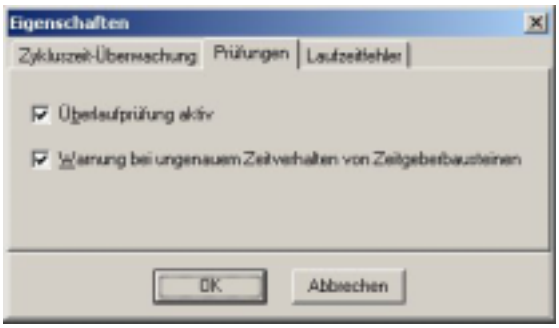
Zeitgeberbausteine können nur funktionieren, wenn die Variable für die anzugebende Zeitdauer PT größer ist als die Abtastschrittweite des Systems unter BORIS. Diese Überprüfung findet immer statt, unabhängig von irgendwelchen Einstellungen. Zusätzlich treten jedoch Ungenauigkeiten auf, die unter Umständen zu Problemen führen können.

Eine Ungenauigkeit tritt immer dann auf, wenn die in PT enthaltene Zeitdauer nicht ganzzahlig durch die Abtastschrittweite teilbar ist. Damit Sie vor Überraschungen verschont bleiben, können Sie bestimmen, ob Sie in einem solchen Fall eine Meldung erhalten möchten. Diese würden Ihnen wie folgt erscheinen:



Meldung bei Überprüfung der Einhaltung der Zeitdauer von Zeitgeberbausteinen (Datei: TIMER.BSY)

Die Einstellung für diese Überprüfung nehmen Sie unter START|EINSTELLUNGEN auf der Registerkarte PRÜFUNGEN vor.



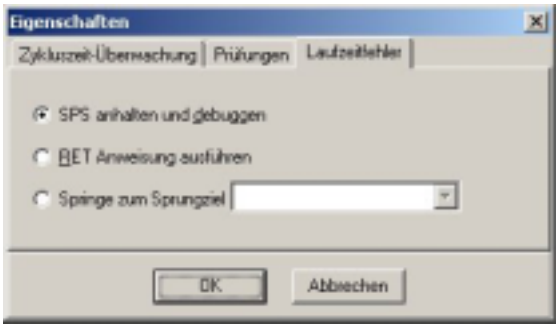
Einstellungen für Überlaufprüfung und Einhaltung der Zeitdauer bei Zeitgeberbausteinen

7.3 Verhalten bei Laufzeitfehlern

Sie können bestimmen, wie die Soft-SPS sich beim Auftreten eines Laufzeitfehlers verhalten soll. Es stehen Ihnen drei Möglichkeiten zur Verfügung:

- Sie können die SPS anhalten und die AWL debuggen,
- direkt aus der AWL mit Hilfe einer RET Anweisung springen oder
- zu einem selbst definierten Sprungziel springen und dort die AWL fortsetzen.

Der unten stehende Dialog bietet Ihnen die obigen Möglichkeiten zu Auswahl an.



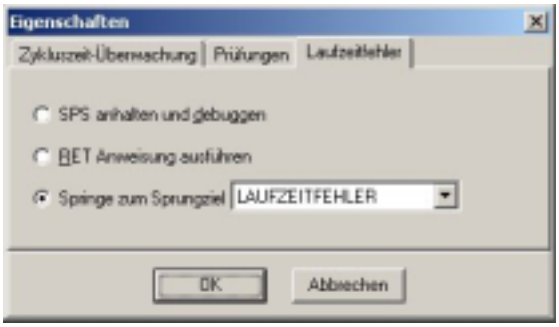
Einstellungen zur Behandlung von Laufzeitfehlern

Voreingestellt ist die Einstellung *SPS anhalten und debuggen*, mit der Sie schnell Laufzeitfehler auffinden und beheben können. Die Einstellung *RET Anweisung ausführen* ist für Fälle geeignet, bei denen Sie nicht zwingend irgendwelche Aktionen durchführen müssen. Sollte es notwendig sein, Ausgänge in einen bestimmten Zustand zu bringen, so können Sie die Auswahl *Springe zum Sprungziel* wählen und an dem angegebenen Sprungziel entsprechende Anweisungen in die AWL einfügen. Die Sprungziele werden von einer vorangehenden Kompilierung (diese erfolgt automatisch, wenn Sie diesen Dialog aufrufen) in die Auswahlliste eingetragen.

- Sie sollten Sprünge bei Laufzeitfehlern nicht bewusst zur *normalen* Programmausführung einsetzen! Gute Programme fangen kritische Fälle wie eine Division durch Null durch eine Abfrage ab!
- Jeder Laufzeitfehler verursacht immer eine Meldung im Meldungsfenster der Entwicklungsumgebung und gegebenenfalls, wenn das Fenster der Umgebung nicht sichtbar ist, im globalen Meldungsfenster.

Nachfolgend soll die Einstellung *Springe zum Sprungziel* an einem Beispiel beschrieben werden. Die nachfolgende Grafik zeigt den Dialog mit der für das Beispiel (*SprungBeiLaufzeitfehler.BSY*) getroffenen

Einstellung bezüglich des Laufzeitverhaltens.



Einstellung des Verhaltens bei Laufzeitfehler

Als Aufgabe soll eine einfache, aber fehlerbehaftete Berechnung dienen. In dieser soll das Integral der Quadratwurzel am Ausgang gebildet werden. Dabei bleiben negative Eingangswerte bewusst unberücksichtigt und führen somit zu einem Laufzeitfehler. Dieser wird nach obiger Einstellung so behandelt, dass die AWL in der Zeile des Sprungziels `LAUFZEITFEHLER` weiterverarbeitet wird. Hier wird nun mit Hilfe eines Zeitgeberbausteins eine bestimmte Zeit gewartet, nach der man davon ausgehen kann, dass keine negativen Werte am Eingang mehr anstehen und die normale Berechnung fortgesetzt werden kann.

```
1   (*
2      Laufzeitfehlerbehandlung durch Angabe eines Sprungziels
3
4      Das Programm soll die Quadratwurzel der Eingangwerte aufsummieren
5      Der Laufzeitfehler tritt bei negativen Eingangswerten auf.
6
7      Es soll dann 5s gewartet werden, da man annimmt, dass dann wieder
8      positive Werte vorhanden sind.
9   *)
10
11
12  VAR_INPUT
13    E_Wert  : REAL;
14  END_VAR
15
16  VAR_OUTPUT
17    A_Wert  : REAL := 0;
18  END_VAR
19
20  VAR
21    init       : BOOL := true;  (* Flag für den Initialisierungslauf           *)
22    LzfB_aktiv : BOOL := false; (* Flag für einen aufgetretenen Laufzeitfehler *)
23    WARTEN     : TP;            (* Funktionsbaustein zur Realisierung der
24                                   Wartezeit                                   *)
25  END_VAR
26
27  VAR CONSTANT
28    WARTEZEIT  : TIME := 2s;
29  END_VAR
30
31  (* Initialisierung erforderlich ?           *)
32                    LD      init
33                    JMPCN   START        (* Wenn nicht Sprung zu START     *)
34
35
36  (*Initialisieren*)
37                    STN     init         (* nur einmalig         *)
38                    LD      WARTEZEIT    (* Wartezeit einstellen *)
39                    ST      WARTEN.PT
40
41
42  (* Prüfen ob Laufzeitfehlerbehandlung aktiv *)
43  START:            LDN     LzfB_aktiv
44                    JMPC    WEITER
45
46
47  (* Laufzeitfehlerbehandlung *)
48  LAUFZEITFEHLER:   LDN     LzfB_aktiv
49                    ST      WARTEN.IN
50                    Cal     WARTEN
51                    LD      WARTEN.Q
52                    ST      LzfB_aktiv
53                    RET
54
55  (* Eigentliches AWL-Programm *)
56  WEITER:           LD      E_Wert
57                    SQRT
58                    ADD     A_Wert
59                    ST      A_Wert
60                    RET
```

Programm zur Demonstration der Laufzeitfehlerbehandlung durch einen Sprung (Datei: Laufzeitfehler.AWL)

Der Laufzeitfehler in der obigen Anweisungsliste tritt in Zeile 57 auf, wenn `E_Wert` negativ wird. Dann wird zur Zeile 48 gesprungen und das Flag `LzfB_aktiv` gesetzt, das erst nach Ablauf der durch `WARTEN.PT` definierten Zeit zurückgesetzt wird.

8 Fehlersuche

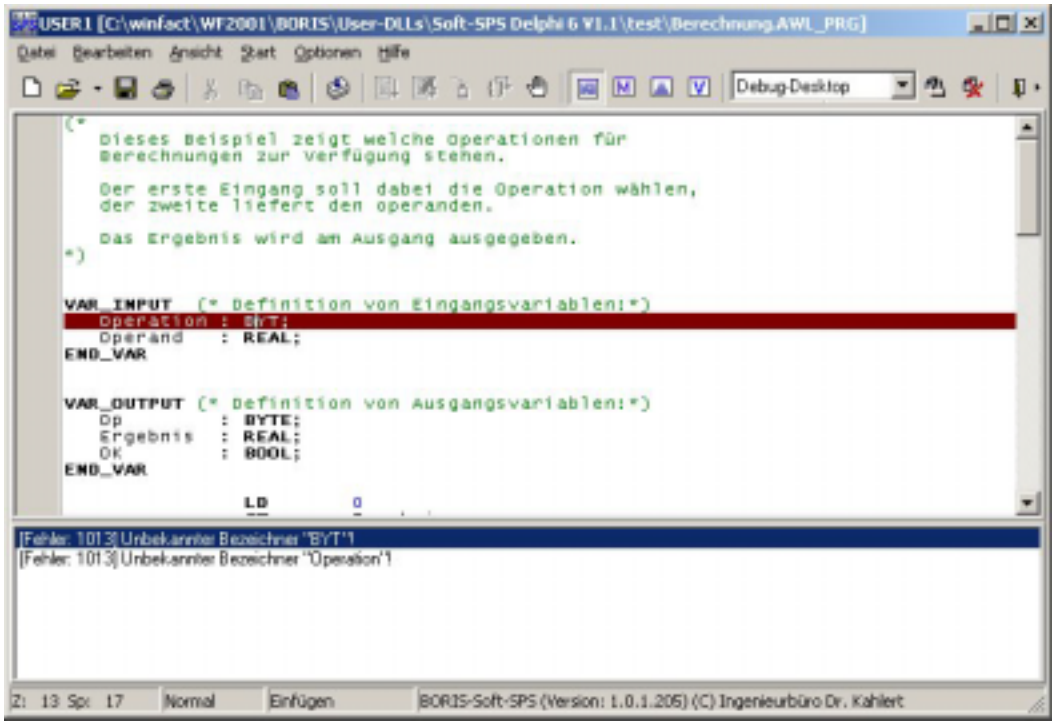
Wesentlich für die Entwicklung von Programmen ist nicht nur ein guter Editor, sondern auch die Möglichkeit zur Fehlersuche. Fehler sind dabei zu unterscheiden in:

- Fehler, die beim Kompilieren auftreten,
- Fehler, die beim Ablauf des AWL-Programms auftreten (Laufzeitfehler) und
- Fehler, die logischer Natur sind (z. B. falscher Programmablauf), d. h. nicht von der Soft-SPS als Fehler erkannt werden können.

8.1 Fehler beim Kompilieren

Wenn das Kompilieren eines SPS-Blockes scheitert, so kann die Simulation nicht gestartet werden. Ein Kompilieren erfolgt immer, wenn die Entwicklungsumgebung geschlossen wird, eine AWL-Datei geladen wird (d.h. es wird auch beim Laden eines BORIS-Simulationssystems kompiliert) oder die Schaltfläche der Werkzeugleiste betätigt wird. Zusätzlich erfolgt beim Simulationsstart eine Überprüfung, ob sich die AWL-Datei geändert hat und eventuell neu kompiliert werden muss.

Treten Fehler beim Kompilieren auf, so werden Sie, falls die Entwicklungsumgebung nicht sichtbar ist, erst beim Starten der Simulation auf einen Fehler aufmerksam gemacht. Nach Öffnen der Entwicklungsumgebung erscheinen im Meldungsfenster dann Informationen bezüglich der Fehler.



Fehlermeldungen des Compilers im Meldungsfenster der Entwicklungsumgebung

Im Meldungsfenster kann durch einen Doppelklick auf die Fehlermeldung zur entsprechenden Position im Quelltext gesprungen werden. Häufig zieht ein Fehler weitere als Konsequenz mit sich: Im obigen Bild ist zu sehen, dass bei der Variablendeklaration des Bezeichners `Operation` ein Fehler auftrat. Demzufolge ist dieser Bezeichner dem Compiler nicht bekannt und eine weitere Fehlermeldung tritt bei Verwendung des Bezeichners auf.

8.2 Laufzeitfehler

Die Soft-SPS bietet Ihnen wesentliche Unterstützung bei der Behandlung von Laufzeitfehlern. Um den Grund eines Laufzeitfehler zu ermitteln, verwenden Sie die Einstellung *SPS anhalten und debuggen* (s. Kapitel Verhalten bei Laufzeitfehlern). In diesem Fall wird beim Auftreten eines Laufzeitfehlers die Entwicklungsumgebung sichtbar und die Ausführung der AWL wird angehalten. Die Ausführungsposition

steht nun hinter der Anweisung, die den Laufzeitfehler verursacht hat. Ist die fehlerhafte Anweisung die letzte in der AWL, so steht die Ausführungsposition auf der ersten Anweisung. Sie können sich nun den Verarbeitungsstapel und Variablen ansehen (s. Kapitel Verarbeitungsstapel und Variablen beobachten/verändern), um eine genaue Analyse des Fehlers vorzunehmen.

8.3 Logische Fehler

Diesen Fehlern ist nur mit der Verwendung von Haltepunkten und der schrittweisen Ausführung des Programms beizukommen. Dabei können Sie mit Hilfe der Anzeige von Variablen, Verarbeitungsstapel und Merkerspeicher logische Fehler leichter finden.

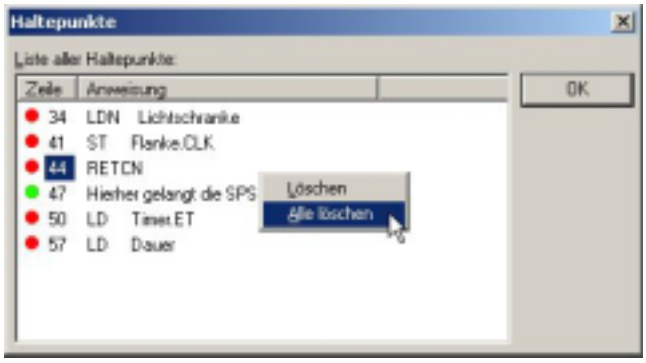
8.3.1 Haltepunkte setzen und löschen

Das Setzen und Löschen von Haltepunkten kann auf verschiedene Weisen geschehen.

1. Mit der Maus:
   Bewegen Sie den Mauszeiger auf den linken breiten Rand des Editors in Höhe der Zeile, auf die der Haltepunkt gesetzt werden soll und betätigen Sie dann die linke Maustaste.
2. Mit der Tastatur:
   Bewegen Sie den Eingabecursor in die Zeile in der ein Haltepunkt gesetzt/gelöscht werden soll und drücken Sie die Tastenkombination Strg + B. Sie können auch den Menüpunkt BEARBEITEN|HALTEPUNKT SETZEN/LÖSCHEN oder die Schaltfläche wählen – die andere Variante ist aber schneller.

Haltepunkte haben keine Wirkung, wenn sie auf eine Zeile gesetzt werden, die keinen Code enthält. In diesem Fall erscheint der Haltepunkt grün statt rot. Sollte der Haltepunkt auf einer Zeile stehen, die eine Anweisung enthält und trotzdem grün erscheinen, so betätigen Sie die Compilieren-Schalfläche der Werkzeugleiste. Anschließend ist der Haltepunkt gültig und rot.

Zusätlich ist es möglich, mehrere Haltepunkte auf einmal zu löschen. Hierzu wählen Sie den Menüpunkt BEARBEITEN|HALTEPUNKTE.... Sie können nun die einzelnen Haltepunkte auswählen, wobei der Editor im Hintergrund den Quelltext um den von Ihnen gewählten Haltepunkt anzeigt. Bei Betätigung der rechten Maustaste in der Liste aller Haltepunkte erscheint ein entsprechendes Popupmenü.

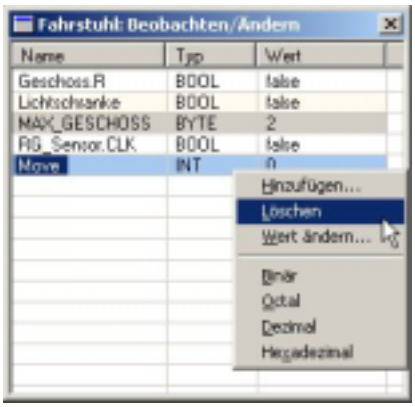


Löschen von mehreren Haltepunkten

8.3.2 Variablen beobachten/verändern

Für eine schnelle Auswertung einzelner Variablen genügt es, während der Ausführung der AWL den Mauszeiger auf den Namen der Variablen im Quelltext zu bewegen und dort kurz stehen zu lassen. Für die Beobachtung über mehrere Anweisungen empfiehlt sich aber die Einrichtung des Beobachtungsfensters. Zum Öffnen dieses Fensters betätigen Sie den Menüpunkt ANSICHT|VARIABLENANZEIGE oder die Schaltfläche . Daraufhin erscheint ein Dialog, in dem Sie durch Betätigung der rechten Maustaste (über ein

Popup-Menü) Variablen hinzufügen, löschen, ändern und das Anzeigeformat einer Variablen wählen können.



Beobachtungsfenster für Variablen

Wählen Sie HINZUFÜGEN..., so erscheint ein weiterer Dialog, in dem eine oder auch mehrere Variablen ausgewählt werden können. Beim Verlassen des Dialoges über die Schaltfläche OK werden die ausgewählten Variablen im Beobachtungsfenster eingetragen.

Sie können das Anzeigeformat einer oder mehrerer Variablen ändern, indem Sie den entsprechenden Menüpunkt des oben gezeigten Popupmenüs wählen. Diese Möglichkeit besteht nur, wenn keine der ausgewählten Variablen vom Typ `BOOL`, `REAL` oder `TIME` ist. Wenn Sie den Wert *einer* Variablen in einem anderen Format als dem eingestellten sehen möchten, genügt es, den Mauszeiger auf den Variablennamen in diesem Fenster zu bewegen und dort kurz stehen zu lassen. Ein kleines Hinweisfenster zeigt Ihnen dann den Wert in allen anderen Formaten an (diese Möglichkeit ist bei den Datentypen `BOOL`, `REAL` und `TIME` nicht gegeben).

Möchten Sie von einer Variablen den WERT ÄNDERN..., so können Sie dieses in einem separaten Dialog:

Ändern des Wertes einer Variablen

Die Werte können hier entsprechend der lexikalischen Konstanten (vgl. Kapitel Lexikalische Konstanten) eingegeben werden. Es lassen sich nur Werte von Variablen und nicht von Konstanten oder Eingangsvariablen ändern.

Konstanten werden wie im Bild *Beobachtungsfenster* bei `MAX_GESCHOSS` auf grauem, Eingangsvariablen auf hellgelbem und Ausgangsvariablen auf hellblauem Hintergrund dargestellt.

8.3.3 Merkerspeicher anzeigen und verändern

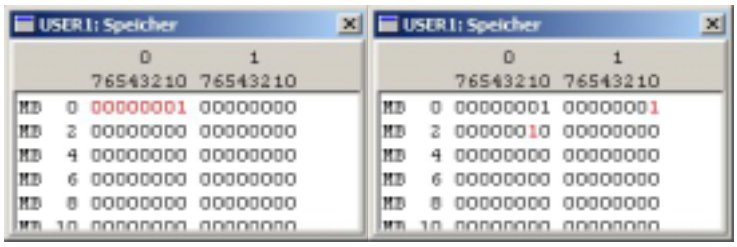
Die Anzeige des Merkerspeichers zeigt zusätzlich zum Inhalt des Speichers die an ihm durchgeführten Änderungen an. Sie können so schnell Änderungen zwischen zwei Haltepunkten ausfindig machen. Das folgende Fenster zeigt den Merkerspeicher nach der Ausführung von Anweisungen zwischen zwei Haltepunkten:

```
1           LD 1
2           ST %MB0
3           ST %M1.0
4           ST %M2.1
5           LD 5
6           ST %M5.2
7           LD 0
```

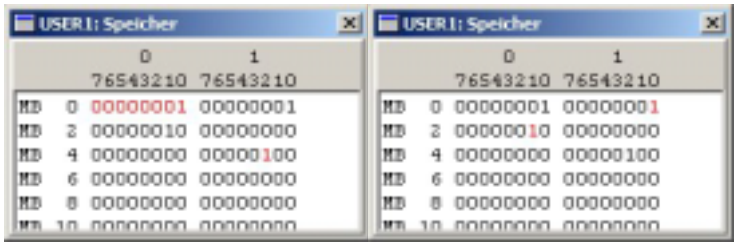
Quelltext zur Betrachtung von Veränderungen am Merkerspeicher
(grau unterlegte Zeilen repräsentieren Haltepunkte)

Da der Merkerspeicher zu Beginn einer Simulation auf 0 initialisiert wird, gibt der erste Durchlauf ein anderes Abbild des Speichers als alle folgenden.

> Die farbliche Hervorhebung von Änderungen existiert nicht dauerhaft. Sobald der Merkerspeicher neu dargestellt werden muss (weil beispielsweise die Größe des Fensters verändert wurde), gehen die Hervorhebungen verloren.



Merkerspeicher beim ersten Durchlauf:
links beim ersten Haltepunkt, rechts beim zweiten Haltepunkt



Merkerspeicher ab dem zweiten Durchlauf:
links beim ersten Haltepunkt, rechts beim zweiten Haltepunkt

Die Anzeige kann wahlweise binär oder hexadezimal erfolgen; Sie wechseln das Anzeigeformat durch Betätigung der rechten Maustaste innerhalb des Anzeigebereichs.

Der Merkerspeicher kann auch direkt in seinem Anzeigefenster verändert werden. Sie können die Schreibmarke innerhalb des Fensters wie in einem Texteditor bewegen. Das Überschreiben einer speziellen Speicherzelle erfolgt durch die direkte Eingabe der entsprechenden Zahl (0 oder 1 für die binäre Darstellung, 0-9 und a-f bzw. A-F für die hexadezimale Darstellung).

8.3.4 Tipps zur Fehlersuche

Manchmal ist es zweckmäßig, einen bedingten Haltepunkt zu definieren. Da diese zurzeit noch nicht unterstützt werden, soll hier gezeigt werden, wie schnell diese nachgebildet werden können. Denkbar wäre zum Beispiel ein Haltepunkt, der erst bei dem 100sten Zyklus wirkt (Beispiel *Bedingter Halt.BSY*).

```
1   (*
2      Dieses Beispiel verdeutlicht die Definition
3      von Eingängen und Ausgängen
4   *)
5
6   VAR_INPUT  (* Definition von Eingangsvariablen:*)
7      INPUT1 : REAL;
8   END_VAR
9
10
11  VAR_OUTPUT (* Definition von Ausgangsvariablen:*)
12     OUTPUT1 : REAL;
13  END_VAR
14
15              (* bedingter Haltepunkt nach dem 100sten Zyklus *)
16               LD       _CYCLE
17               EQ       100
18               JMPCN    WEITER
19               NOT      (* diese Anweisung wird nur für den Haltepunkt benötigt *)
20              (* ursprüngliches Programm: *)
21  WEITER:      LD       INPUT1
22               MUL      2.5
23               ST       OUTPUT1
24               RET      (* ist am Ende einer AWL nicht zwingend *)
```

Beispiel für einen bedingten Haltepunkt

Wie leicht zu erkennen ist, werden vier Anweisungen benötigt, um den „bedingten Haltepunkt“ setzen zu können. Zwei, die in Zeile 16 und 17, bilden die Bedingung. Im Anschluss daran erfolgt ein bedingter Sprung (dritte Anweisung) über die vierte Anweisung, auf der der Haltepunkt gesetzt ist. Diese wird also nur unter der Bedingung `_CYCLE = 100` erreicht.

9 Anweisungslisten verwalten

Zu den Verwaltungsaufgaben gehören das Laden und Speichern von AWL, ebenso wie das Drucken und Exportieren.

9.1 Laden und Speichern

Eine Anweisungliste kann über die Menüpunkte DATEI|ÖFFNEN... geladen und über DATEI|SPEICHERN bzw., wenn noch kein Name existiert, über DATEI|SPEICHERN UNTER... gespeichert werden. Entsprechend stehen auf der Werkzeugleiste für das Laden und Speichern die gängigen Symbole und zur Verfügung. Rechts neben der Schaltfläche zum Öffnen einer Anweisungsliste ist ein kleiner Pfeil, der nach unten zeigt. Beim Drücken des Pfeils klappt ein Menü mit den von Ihnen zuletzt geöffneten Anweisungslisten auf (im Menü DATEI werden diese ebenfalls eingetragen).

9.2 Exportieren

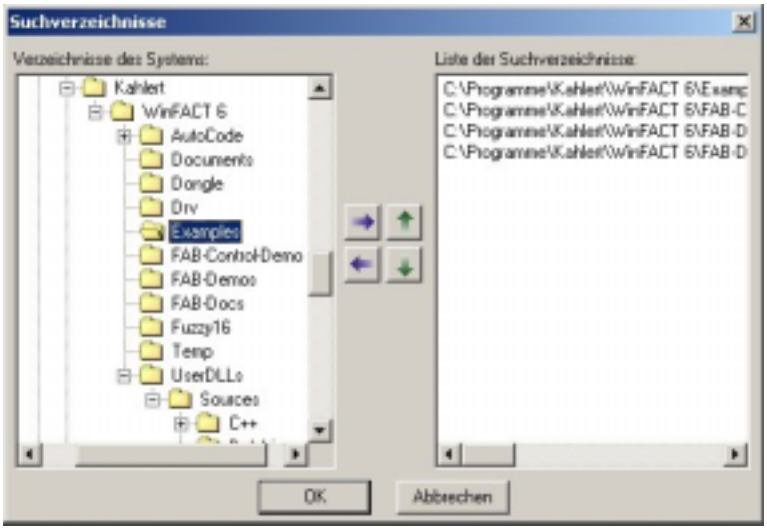
Damit Sie die AWL in syntaktisch hervorgehobenen Farben und in dem entsprechenden Zeichensatz in andere Dokumente einbinden können, steht Ihnen der Menüpunkt DATEI|RTF-EXPORT... zur Verfügung (so wurden unter anderem die Listings in diesem Dokument erzeugt).

9.3 Drucken

Das Drucken erfolgt durch Betätigung des Menüpunktes DATEI|DRUCKEN..., worauf der Standarddialog zum Einstellen des Druckers angezeigt wird. Die entsprechende Schaltfläche der Werkzeugleiste bietet Ihnen die gleiche Funktionalität.

9.4 Suchverzeichnisse

In den BORIS-Systemdateien wird keine AWL, sondern nur der Dateiname der AWL hinterlegt. Wenn Sie nun ein System mit einem Soft-SPS-Block auf einen anderen Rechner übertragen möchten, so kann es vorkommen, dass die AWL-Datei nicht gefunden werden kann, weil beispielsweise eine andere Verzeichnisstruktur auf diesem Rechner vorliegt. Um diesen Missstand möglichst einfach entgegen zu wirken, können Sie die Verzeichnisse, in denen sich die AWL-Dateien befinden, in die Liste der Suchverzeichnisse eintragen. Suchverzeichnisse werden über den Menüpunkt OPTIONEN|SUCHVERZEICHNISSE... hinzufügen verwaltet.



Dialog zum Verwalten der Suchverzeichnisse

Wird eine AWL-Datei eines Blockes nicht gefunden, so werden die in diesem Dialog angegebenen Suchverzeichnisse von oben nach unten durchsucht. Die Suche bricht ab, sobald die Datei gefunden wird. Daher ist die Reihenfolge der Suchverzeichnisse von Bedeutung, die durch die grünen Pfeiltasten verändert werden kann, wenn ein Suchverzeichnis aus der Liste ausgewählt ist.

10 Anhang

10.1 AWL-Befehle

Die Befehle der *WINFACT*-Soft-SPS wurden in Anlehnung an die internationale Norm IEC 61131-3 erstellt. Die hier verwendete Beschreibung stützt sich auf die Begriffe des Verarbeitungsstapels und des aktuellen Ergebnisses *AE* (s. Kapitel Verarbeitungsstapel). Folgende Befehle werden unterstützt:

LD

Verwendung:	`LD X`
Bedeutung:	Laden (load)
Beschreibung:	Überschreibt Typ und Wert des *AE* mit denen von `X`.
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: +, TIME: +

AE nach Ausführung:	Enthält eine Kopie von X.
Mögliche Ausnahmen:	-

LDN

Verwendung:	LDN X
Bedeutung:	Lade negiert (load negated)
Beschreibung:	Überschreibt den Typ des *AE* mit dem von X und lädt den negierten Wert von X in das *AE*.
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: -, TIME: -
AE nach Ausführung:	Entspricht dem Typ und enthält den negierten Wert von X.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation

ST

Verwendung:	ST X
Bedeutung:	Speichern (store)
Beschreibung:	Speichert den Wert des *AE* in X. Dabei wird, falls das *AE* und X von unterschiedlichem Typ sind, eine Typumwandlung von dem Typ des *AE* zu dem Typ von X vorgenommen (s. Kapitel *Automatische Typkonvertierung*). Hierbei kann eine Bereichsüberschreitung eintreten, wenn die entsprechende Prüfung eingeschaltet ist.
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: +, TIME: +
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	• Der Typ *A* kann nicht in den Typen *B* konvertiert werden. • Fehler bei Bereichsüberprüfung

STN

Verwendung:	STN X
Bedeutung:	Speichere negiert (store negated)
Beschreibung:	Speichert den negierten Wert des *AE* in X. Dabei wird, falls das *AE* und X von unterschiedlichem Typ sind, eine Typumwandlung von dem Typ des *AE* zu dem Typ von X vorgenommen (s. Kapitel *Automatische Typkonvertierung*). Hierbei kann eine Bereichsüberschreitung eintreten, wenn die entsprechende Prüfung eingeschaltet ist.
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: -, TIME: -

AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	• Der Typ *A* kann nicht in den Typen *B* konvertiert werden • Fehler bei Bereichsüberprüfung • Falscher Typ für diese Operation

STC

Verwendung:	`STC X`
Bedeutung:	Bedingtes Speichern (store conditional true)
Beschreibung:	Speichert `TRUE` in `X`, wenn eine boolesche Auswertung des *AEs* zu `TRUE` führt. Implizit wird eine Typumwandlung von dem Typ des *AE* zu dem `BOOL` vorgenommen, um die boolesche Auswertung durchzuführen (s. Kapitel *Automatische Typkonvertierung*).
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: +, TIME: +
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

STCN

Verwendung:	`STCN X`
Bedeutung:	Bedingtes Speichern bei `FALSE` (store conditional not true)
Beschreibung:	Speichert `FALSE` in `X`, wenn eine boolesche Auswertung des *AEs* zu `TRUE` führt. Implizit wird eine Typumwandlung von dem Typ des *AE* zu dem `BOOL` vorgenommen, um die boolesche Auswertung durchzuführen (s. Kapitel *Automatische Typkonvertierung*).
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: +, TIME: +
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

JMP

Verwendung:	`JMP` *`Sprungmarke`*
Bedeutung:	Springe (jump)
Beschreibung:	Springt in die Zeile der AWL, in der die Sprungmarke angegeben ist.
Operandentypen:	-
AE nach Ausführung:	Bleibt unverändert

Mögliche Ausnahmen:	-

JMPC

Verwendung:	`JMPC` *`Sprungmarke`*
Bedeutung:	Bedingter Sprung (jump conditional if true)
Beschreibung:	Springt in die Zeile der AWL, in der die Sprungmarke angegeben ist, wenn die boolesche Auswertung des *AE* `TRUE` ergibt (s. Kapitel *Automatische Typkonvertierung*).
Operandentypen:	-
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

JMPCN

Verwendung:	`JMPCN` *`Sprungmarke`*
Bedeutung:	Bedingter Sprung bei `FALSE` (jump conditional if not true)
Beschreibung:	Springt in die Zeile der AWL, in der die Sprungmarke angegeben ist, wenn die boolesche Auswertung des *AE* `FALSE` ergibt (s. Kapitel *Automatische Typkonvertierung*).
Operandentypen:	-
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

JMPL

Verwendung:	`JMPL`
	`JMP  Ziel0`
	`JMP  Ziel1`
	…
	`JMP  ZielN`

Bedeutung:	Springe in Sprungliste (JuMP into List of jumps)
Beschreibung:	Durch diese Anweisung wird die Möglichkeit geschaffen, aufgrund eines Wertes unterschiedlich zu verzweigen. Die `JMPL` Anweisung wandelt zunächst den Typ des *AE* in einen `DWORD` um (ohne Bereichsüberprüfung). Der so gebildete Wert ist größer oder gleich `0` und gibt die Anzahl der Anweisungen an, die nach vorn gesprungen werden soll. `0` bedeutet dabei die Ausführung der nächsten Anweisung (also `JMP Ziel0`), `1` die der übernächsten (also `JMP Ziel1`) usw.. Der Wert für die nach vorn zu springenden Anweisungen wird auf N

	begrenzt. Dies bedeutet, dass in allen Fällen, in denen *AE* größer oder gleich *N* ist, die Zeile `JMP  ZielN` ausgeführt wird. Vom Compiler wird erwartet, dass direkt nach `JMPL` mindestens eine `JMP`-Anweisung stehen muss. Somit steht diese Anweisung immer im Verbund nachfolgender `JMP`-Anweisungen. Dieser Verbund bricht an der ersten von `JMP` unterschiedlichen Anweisung ab.
Operandentypen:	
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

RET

Verwendung:	`RET`
Bedeutung:	Rücksprung (return)
Beschreibung:	Beendet die Ausführung der AWL für den aktuellen Simulationsschritt.
Operandentypen:	-
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

RETC

Verwendung:	`RETC`
Bedeutung:	Bedingter Rücksprung (return conditional if true)
Beschreibung:	Beendet die Ausführung der AWL für den aktuellen Simulationsschritt, wenn die boolesche Auswertung des *AE* `TRUE` ergibt (s. Kapitel *Automatische Typkonvertierung*).
Operandentypen:	-
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

RETCN

Verwendung:	`RETCN`
Bedeutung:	Bedingter Rücksprung bei `FALSE` (return conditional if not true)
Beschreibung:	Beendet die Ausführung der AWL für den aktuellen Simulationsschritt, wenn die boolesche Auswertung des *AE* `FALSE` ergibt (s. Kapitel *Automatische Typkonvertierung*).

Operandentypen:	-
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	-

CAL

Verwendung:	`CAL` *`Funktionsbausteininstanz`*
Bedeutung:	Aufruf eines Funktionsbausteines (call)
Beschreibung:	Ruft den Funktionsbaustein, der durch den Typ der Funktionsbausteininstanz definiert ist, auf. Eine Funktionsbausteininstanz ist dabei eine Variable vom Typ eines Funktionbausteins. Der Aufruf hat zur Folge, dass die Berechnungsvorschrift für den entsprechenden Baustein durchgeführt wird. Dabei werden die Eingangsvariablen ausgewertet und die Ausgangsvariablen definiert. Eine detaillierte Beschreibung findet sich im Kapitel *Standard-Funktionsbausteine*.
Operandentypen:	Funktionsbausteininstanz
AE nach Ausführung:	Beliebig (d. h. Sie sollten als erste Anweisung nach dieser *`LD`* oder *`LDN`* verwenden.)
Mögliche Ausnahmen:	Für Zeitgeberbausteine (s. Kapitel *Zeitgeber*) kann eine Abarbeitung nur erfolgen, wenn die Abtastschrittweite von BORIS kleiner ist als die Zeitdauer der Bausteinvariablen `PT.` Daher können Zeitgeberbausteine die folgende Ausnahme auslösen: • Der Funktionsbaustein *A* kann mit der eingestellten Abtastschrittweite von BORIS nicht arbeiten! Des Weiteren können Zeitgeberbausteine eine Warnung auslösen (s. Kapitel *Warnung bei ungenauem Zeitverhalten von Zeitgeberbausteinen*).

CALC

Verwendung:	`CALC` *`Funktionsbausteininstanz`*
Bedeutung:	Bedingter Aufruf eines Funktionsbausteines
Beschreibung:	Ruft den Funktionsbaustein, der durch den Typ der Funktionsbausteininstanz definiert ist, auf, wenn die boolesche Auswertung des *AE* `TRUE` ergibt. Vergleiche *`CAL`*
Operandentypen:	Funktionsbausteininstanz
AE nach Ausführung:	Siehe *`CAL`*
Mögliche Ausnahmen:	Siehe *`CAL`*

CALCN

Verwendung:	`CALCN` *`Funktionsbausteininstanz`*
Bedeutung:	Bedingter Aufruf eines Funktionsbausteines bei `FALSE`
Beschreibung:	Ruft den Funktionsbaustein, der durch den Typ der Funktionsbausteininstanz definiert ist, auf, wenn die boolesche Auswertung des *AE* `FALSE` ergibt. Vergleiche *`CAL`*
Operandentypen:	Funktionsbausteininstanz
AE nach Ausführung:	Siehe *`CAL`*
Mögliche Ausnahmen:	Siehe *`CAL`*

AND, OR, XOR

Verwendung:	`AND X`
	`OR X`
	`XOR X`

Bedeutung:	Verknüpfungen: UND ODER EXCLUSIV ODER
Beschreibung:	`X` und das *AE* werden miteinander durch die angegebene Operation verknüpft. Zuvor wird der kleinste Typ bestimmt, der die Werte beider Operanden aufnehmen kann (s. Kapitel *Automatische Typkonvertierung*). Anschließend werden beide Werte in den so bestimmten Typ ohne Bereichsüberprüfung gewandelt und die Verknüpfungsoperation durchgeführt.
Operandentypen:	BOOL: +, BYTE: +, WORD: +, DWORD: +, INT: +, DINT: +, REAL: -, TIME: -
AE nach Ausführung:	Haben *AE* und `X` den gleichen Typ vor der Operation, so ist dies der Typ nach der Operation. Sonst ist der Ergebnistyp der aus der automatischen Typkonvertierung. Der Wert entspricht dem Ergebnis der Operation.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation

NOT

Verwendung:	`NOT`
Bedeutung:	Nicht
Beschreibung:	Negiert den Inhalt des AE. Bei dieser Operation werden alle Bits des AE umgekehrt: die, die `1` waren, werden zu `0` und die, die `0` waren, werden zu `1`.
Operandentypen:	BOOL, BYTE, WORD, DWORD, INT, DINT, REAL, TIME

	+ + + + + + - -
AE nach Ausführung:	Der Typ bleibt erhalten.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation

ANDN, ORN, XORN

Verwendung:	`ANDN X`
	`ORN X`
	`XORN X`

Bedeutung:	Verknüpfungen: UND NICHT ODER NICHT EXCLUSIV ODER NICHT
Beschreibung:	Wie bei *AND, OR, XOR*, jedoch mit zuvor negiertem X.
Operandentypen:	BOOL BYTE WORD DWORD INT DINT REAL TIME + + + + + + - -
AE nach Ausführung:	Haben *AE* und X den gleichen Typ vor der Operation, so ist dies der Typ nach der Operation. Sonst ist der Ergebnistyp der aus der automatischen Typkonvertierung. Der Wert entspricht dem Ergebnis der Operation.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation

SHL, SHR

Verwendung:	`SHL X` `SHR X`
Bedeutung:	Bitweises Schieben nach links (shift left) Bitweises Schieben nach rechts (shift right)
Beschreibung:	Schiebt den Wert des *AE* um die Anzahl der durch X angegebenen Bits nach links bzw. rechts. Folgende Typen sind für *AE* zulässig: BOOL BYTE WORD DWORD INT DINT REAL TIME - + + + + + - -
Operandentypen:	BOOL BYTE WORD DWORD INT DINT REAL TIME - + - - - - - -
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	• Falscher Typ für diese Operation (meint in diesem Fall immer den Typ des *AE*; der Operandentyp wird beim Kompilieren geprüft.)

ROL, ROR

Verwendung:	`ROL X`

	ROR X
Bedeutung:	Bitweises Rotieren nach links (rotate left) Bitweises Rotieren nach rechts (rotate right)
Beschreibung:	Rotiert den Wert des *AE* bitweise um die Anzahl der durch X angegebenen Bits links bzw. rechts herum. Folgende Typen sind für *AE* zulässig: BOOL: - | BYTE: + | WORD: + | DWORD: + | INT: + | DINT: + | REAL: - | TIME: -
Operandentypen:	BOOL: - | BYTE: + | WORD: - | DWORD: - | INT: - | DINT: - | REAL: - | TIME: -
AE nach Ausführung:	Bleibt unverändert
Mögliche Ausnahmen:	• Falscher Typ für diese Operation (meint in diesem Fall immer den Typ des *AE;* der Operandentyp wird beim Kompilieren geprüft.)

LT, LE, EQ, NE, GE, GT

Verwendung:	LT X LE X EQ X NE X GE X GT X
Bedeutung:	Vergleichsoperationen: *AE* < X (less) *AE* <= X (less or equal) *AE* = X (equal) *AE* ≠ X (not equal) *AE* >= X (greater or equal) *AE* > X (greater)
Beschreibung:	Führt die entsprechende Vergleichsoperation zwischen dem Wert des *AE* und dem des Operanden X durch. Zunächst wird geprüft, ob einer der Operanden vom Typ TIME ist. In diesem Fall muss der andere ebenfalls diesen Typ besitzen. Bei allen anderen Typen wird der kleinste Typ bestimmt, der beide Werte aufnehmen kann (s. Kapitel *Automatische Typkonvertierung*) und falls erforderlich eine Konvertierung beider Operanden (*AE* und X) durchgeführt. Die Typkonvertierung erfolgt gegebenenfalls mit Bereichsüberprüfung (s. Kapitel *Überlaufprüfung*).
Operandentypen:	BOOL | BYTE | WORD | DWORD | INT | DINT | REAL | TIME

	+ + + + + + + *AE*=TIME
AE nach Ausführung:	BOOL mit dem Ergebnis des Vergleichs
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Fehler bei Bereichsüberprüfung

ADD, SUB

Verwendung:	ADD X SUB X
Bedeutung:	Addition (addition) Subtraktion (subtraction)
Beschreibung:	Führt die entsprechende Operation zwischen dem Wert des *AE* und dem des Operanden X durch. Dabei wird gegebenenfalls eine Bereichsüberprüfung durchgeführt (s. Kapitel *Überlaufprüfung*). Zunächst wird geprüft, ob einer der Operanden vom Typ TIME ist. In diesem Fall muss der andere ebenfalls diesen Typ besitzen. Bei allen anderen Typen wird der kleinste Typ bestimmt, der beide Werte aufnehmen kann (s. Kapitel *Automatische Typkonvertierung*) und falls erforderlich eine Konvertierung beider Operanden (*AE* und X) durchgeführt. Die Typkonvertierung erfolgt gegebenenfalls mit Bereichsüberprüfung.
Operandentypen:	BOOL: - ; BYTE: + ; WORD: + ; DWORD: + ; INT: + ; DINT: + ; REAL: + ; TIME: *AE*=TIME
AE nach Ausführung:	Haben *AE* und X den gleichen Typ vor der Operation, so ist dies der Typ nach der Operation. Sonst ist der Ergebnistyp derjenige, der beide Operanden (X und *AE*) aufnehmen konnte. Der Wert entspricht dem Ergebnis der Operation.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Fehler bei Bereichsüberprüfung

MUL

Verwendung:	MUL X
Bedeutung:	Multiplikation (multiplication)
Beschreibung:	Führt die entsprechende Operation zwischen dem Wert des *AE* und dem des Operanden X durch. Dabei wird gegebenenfalls eine Bereichsüberprüfung durchgeführt (s. Kapitel *Überlaufprüfung*). Zunächst wird geprüft, ob einer der Operanden vom Typ TIME ist. In diesem Fall darf der andere diesen Typ **nicht** besitzen. Ist keiner der beiden Operanden vom Typ TIME, so wird der kleinste Typ bestimmt, der beide Werte aufnehmen kann (s. Kapitel *Automatische Typkonvertierung*) und überprüft, ob sich der so festgestellte Typ vergrößern lässt, um das Ergebnis der Operation

	aufzunehmen. Dabei wird folgende Tabelle verwendet: <table><tr><td>Typ nach automatischer Typkonvertierung</td><td>vergrößerter Typ</td></tr><tr><td>BYTE</td><td>WORD</td></tr><tr><td>WORD</td><td>DWORD</td></tr><tr><td>INT</td><td>DINT</td></tr><tr><td>Alle anderen…</td><td>…bleiben gleich</td></tr></table> Der auf diese Weise ermittelte Typ wird für *AE* verwendet.
Operandentypen:	<table><tr><td>BOOL</td><td>BYTE</td><td>WORD</td><td>DWORD</td><td>INT</td><td>DINT</td><td>REAL</td><td>TIME</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td><td>*AE*≠TIME</td></tr></table>
AE nach Ausführung:	Der Ergebnistyp ist wie nach obiger Beschreibung ein vergrößerter Typ einer automatischen Typkonvertierung. Der Wert entspricht dem Ergebnis der Operation.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Fehler bei Bereichsüberprüfung

DIV

Verwendung:	DIV X
Bedeutung:	Division (division)
Beschreibung:	Führt die Division *AE*/X durch. X darf nur dann vom Typ TIME sein, wenn *AE* ebenfalls den Typ TIME besitzt.
Operandentypen:	<table><tr><td>BOOL</td><td>BYTE</td><td>WORD</td><td>DWORD</td><td>INT</td><td>DINT</td><td>REAL</td><td>TIME</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td><td>*AE*=TIME</td></tr></table>
AE nach Ausführung:	Falls eine Division zweier Zeiten durchgeführt wurde, ist das Ergebnis vom Typ REAL. Sonst bleibt der Typ von *AE* erhalten.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Fehler bei Bereichsüberprüfung • Division durch Null

MOD

Verwendung:	MOD X
Bedeutung:	Bestimmung des Restes einer Division (modulo division)
Beschreibung:	Berechnet: AE = *AE* - INT(*AE* / X) • X. Zunächst wird geprüft, ob einer der Operanden vom Typ TIME ist. In diesem Fall muss der andere ebenfalls diesen Typ besitzen. Bei allen anderen Typen wird der kleinste Typ bestimmt, der beide

	Werte aufnehmen kann (s. Kapitel *Automatische Typkonvertierung*) und falls erforderlich eine Konvertierung beider Operanden (*AE* und X) durchgeführt. Die Typkonvertierung erfolgt, gegebenenfalls mit Bereichsüberprüfung.
Operandentypen:	BOOL: - ; BYTE: + ; WORD: + ; DWORD: + ; INT: + ; DINT: + ; REAL: + ; TIME: *AE*=TIME
AE nach Ausführung:	Der Typ von *AE* bleibt erhalten
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Fehler bei Bereichsüberprüfung • Division durch Null

ABS

Verwendung:	ABS
Bedeutung:	Betrag von *AE* (absolut)
Beschreibung:	Bestimmt den Betrag des *AE*.
Operandentypen:	BOOL: - ; BYTE: + ; WORD: + ; DWORD: + ; INT: + ; DINT: + ; REAL: + ; TIME: - Die Typen BYTE, WORD und DWORD sind zulässig, aber ohne Wirkung.
AE nach Ausführung:	Der Typ von *AE* bleibt erhalten; der Wert ist positiv oder 0.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation

NEG

Verwendung:	NEG
Bedeutung:	Negiert *AE* (negate)
Beschreibung:	Das Ergebnis der Negation ist immer vorzeichenbehafteter Typ (s. Kapitel *Datentypen*); der Wert entspricht dem der Multiplikation mit -1. Hat *AE* vor der Operation den Typ DWORD, so kann eine Bereichsüberschreitung (wenn die Überprüfung eingestellt ist; s. Kapitel *Überlaufprüfung*) auftreten, da der Zahlenbereich von DWORD nicht vollständig in den größten vorzeichenbehafteten Typen DINT passt.
Operandentypen:	BOOL: - ; BYTE: + ; WORD: + ; DWORD: + ; INT: + ; DINT: + ; REAL: + ; TIME: - Die Typen BYTE, WORD und DWORD sind zulässig, aber ohne Wirkung.
AE nach Ausführung:	Der Typ von *AE* wird zu einem vorzeichenbehafteten Typ.
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Fehler bei Bereichsüberprüfung

LN

Verwendung:	LN
Bedeutung:	Natürlicher Logarithmus (logarithmus naturalis)
Beschreibung:	Berechnet den Logarithmus zur Basis e=2,7182818…
Operandentypen:	BOOL: - / BYTE: + / WORD: + / DWORD: + / INT: + / DINT: + / REAL: + / TIME: -
AE nach Ausführung:	REAL
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Gleitkommadivision durch Null

LOG

Verwendung:	LOG
Bedeutung:	Dekadischer Logarithmus (logarithmus)
Beschreibung:	Berechnet den Logarithmus zur Basis 10
Operandentypen:	BOOL: - / BYTE: + / WORD: + / DWORD: + / INT: + / DINT: + / REAL: + / TIME: -
AE nach Ausführung:	REAL
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Gleitkommadivision durch Null

EXP

Verwendung:	EXP
Bedeutung:	Exponentialfunktion
Beschreibung:	Berechnet den den Wert e^*AE*
Operandentypen:	BOOL: - / BYTE: + / WORD: + / DWORD: + / INT: + / DINT: + / REAL: + / TIME: -
AE nach Ausführung:	REAL
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Gleitkommaüberlauf

SQRT

Verwendung:	SQRT
Bedeutung:	Quadratwurzel (square root)

Beschreibung:	Berechnet die Wurzel aus *AE*.
Operandentypen:	BOOL: - / BYTE: + / WORD: + / DWORD: + / INT: + / DINT: + / REAL: + / TIME: -
AE nach Ausführung:	REAL
Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Ungültige Gleitkommaoperation

SIN, COS, TAN

Verwendung:	SIN COS TAN
Bedeutung:	Sinus Kosinus Tangens
Beschreibung:	Berechnet die entsprechende Winkelfunktion.
Operandentypen:	BOOL: - / BYTE: + / WORD: + / DWORD: + / INT: + / DINT: + / REAL: + / TIME: -
AE nach Ausführung:	REAL
Mögliche Ausnahmen:	• Falscher Typ für diese Operation

ASIN, ACOS, ATAN

Verwendung:	ASIN ACOS ATAN
Bedeutung:	Arcussinus Arcuskosinus Arcustangens
Beschreibung:	Berechnet die entsprechende Umkehrfunktion der Winkelfunktion (gibt den Hauptwert zurück). Die Definitionsbereiche sind wie folgt: ASIN -1..1 ACOS -1..1 ATAN -∞..+∞
Operandentypen:	BOOL: - / BYTE: + / WORD: + / DWORD: + / INT: + / DINT: + / REAL: + / TIME: -
AE nach Ausführung:	REAL

Mögliche Ausnahmen:	• Falscher Typ für diese Operation • Ungültige Gleitkommaoperation (*AE* lag außerhalb Definitionsbereiches)

Operationen mit Klammer

Verwendung:	`AND( X` `ADD( Y` `LT( Z`
	`OR( X` `SUB( Y` `LE( Z`
	`XOR( X` `MUL( Y` `EQ( Z`
	`ANDN( X` `DIV( Y` `NE( Z`
	`ORN( X` `MOD( Y` `GE( Z`
	`XORN( X` `GT( Z`

Bedeutung:	Siehe vorangehende Beschreibung der einzelnen Anweisungen: *AND, OR, XOR*, *NOT*, *ANDN, ORN, XORN*, *ADD, SUB*, *MUL*, *DIV*, *MOD*, *LT, LE, EQ, NE, GE, GT*
Beschreibung:	Zweck der Operationen mit Klammer ist die Ausführung der entsprechenden Operation auf dem *AE* und dem Ergebnis in der Klammer. Die Ausführung der Operation kann daher erst bei der passenden schließenden Klammer (s. u) erfolgen. Daher wird sie zusammen mit dem Operanden auf den Verarbeitungsstapel geschoben (s. Kapitel *Verarbeitungsstapel*). `X` oder `Y`, bzw. `Z` wird so zum *AE*, mit dem nun der innere Teil der Klammer berechnet werden kann.
Operandentypen:	Siehe Beschreibung der einzelnen Anweisungen
AE nach Ausführung:	Entspricht `X` oder `Y`, bzw. `Z`.
Mögliche Ausnahmen:	Siehe Beschreibung der einzelnen Anweisungen

)

Verwendung:	)
Bedeutung:	Schließen der Klammerebene und Ausführen der zugehörigen Operation
Beschreibung:	Nimmt die durch eine der *Operationen mit Klammer* auf den Verarbeitungsstapel (s. Kapitel *Verarbeitungsstapel*) liegende Operation samt Operand vor. Das *AE* wird dabei gleich dem *AE* vor der Operation mit der öffnenden Klammer und die Operation wird mit eben diesem *AE* und dem Operanden vom Stapel durchgeführt.
Operandentypen:	Je nach Zustand des Verabeitungsstapels

AE nach Ausführung:	Je nach Zustand des Verabeitungsstapels
Mögliche Ausnahmen:	Je nach Zustand des Verabeitungsstapels

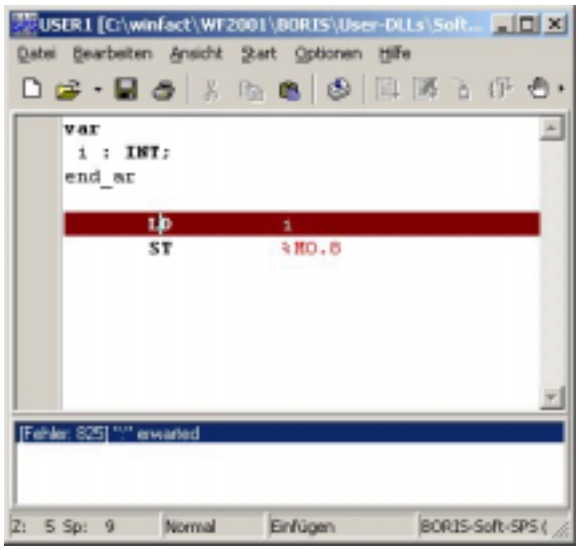
Typumwandlung

Verwendung:	`xxxx_TO_yyyy`
Bedeutung:	Typumwandlung
Beschreibung:	Es gibt folgende Möglichkeiten zur expliziten Typumwandlung: yyyy / xxxx BOOL BYTE WORD DOWRD INT DINT REAL TIME BOOL - + + + + + + BYTE + - + + + + + + WORD + + - + + + + + DWORD + + + - + + + + INT + + + + - + + + DINT + + + + + - - + REAL + + + + + + - + TIME + + + + + + + - Bei allen expliztien Typumwandlungen erfolgt keine Bereichs-überprüfung.
Operandentypen:	siehe Beschreibung
AE nach Ausführung:	Entspricht dem Typ von `yyyy`.
Mögliche Ausnahmen:	• Das aktuelle Ergebnis (*AE*) ist vom Typ *zzzz* statt `xxxx`.

10.2 Compiler-Fehlermeldungen

800-838 und 900-924 *xyz* erwartet

Diese Fehlermeldung deutet auf einen syntaktischen Fehler hin. An der angegebenen Stelle wurde *xyz* als Zeichen(-kette) erwartet. Es kann durchaus vorkommen, dass Ihnen die Position des Fehlers nicht korrekt vorkommt, die Grammatik der AWL jedoch bis zu dieser Stelle korrekt war. Beispiel:



Anscheinend fehlerhafte Position der Fehlermeldung

Der Grund für die im obigen Bild markierte Fehlerposition ist, dass die Zeichenkette `end_ar`, die vermutlich `end_var` heißen sollte, als Variablenbezeichner interpretiert wurde. Dies ist an dieser Stelle korrekt; die Variablendeklaration ist ja noch nicht durch `end_var` beendet worden. Daher wird als nächstes Zeichen (also hinter dem Bezeichner `end_ar`) der Doppelpunkt für die Variablendeklaration erwartet.

Die Position des eigentlichen Fehlers kann also bei diesen Fehlermeldungen nur bedingt ausfindig gemacht werden. Sie liegt aber immer vor der angegebenen Position.

839-841 und 925–927 ungültig *xyz*

Die Fehlermeldungen dieser Kategorie deuten auf eine Zeichenkette hin, die nicht als ein bestimmtes syntaktisches Element interpretiert werden kann. Lesen Sie dazu unter den entsprechenden Kapiteln nach:

LexConst	*Lexikalische Konstanten*
DirectAddr	*Eingebaute Variablen*
Operand	In diesem Fall konsultieren Sie bitte die Hilfe der verwendeten Anweisung (s. *AWL-Befehle*).

1000 Ungültige Referenz zu einem Bit des Merkerspeichers!

Dieser Fehler tritt auf bei der Referenzierung eines Bits des Merkerspeichers durch die Verwendung von `%MX`*`n.m`* bzw. `%M`*`n.m`*. Da durch diese Referenz immer nur ein Bit eines Bytes angesprochen wird, muss `m` im Bereich von `0..7` liegen.

1001 Lexikalische Konstante kann nicht interpretiert werden!

Es trat ein Fehler auf bei dem Versuch, eine lexikalische Konstante auszuwerten. Die Möglichkeiten, eine fehlerhafte lexikalische Konstante anzugeben, sind im Prinzip:

- Bei der Angabe einer Basis muss diese einem der Werte `16`, `10`, `8` oder `2` entsprechen und durch # von dem Wert der Konstante getrennt sein.
- Bei einer Zeitangabe muss die Einheit angegeben werden: `d`, `h`, `m`, `s` oder `ms`.
- Werte für den Typ REAL müssen, falls sie keinen Punkt besitzen, durch eine vorangestellte Typangabe spezifiziert werden, z. B. `REAL#5`.

- Erforderlicher Typ und Konstante passen nicht zu einander.

Näheres unter Lexikalische Konstanten.

1002 Unbekannter Operator *xyz*!

Der Operator bzw. die Anweisung ist dem Compiler nicht bekannt. Korrigieren Sie den Operator resp. die Anweisung.

1003 Der Operator *xzy* mit Operand vom Typ *t* ist unzulässig! Zulässig ist/sind: …!

Die angegebene Anweisung bzw. der verwendete Operator erfordert einen anderen Typ als den in der Fehlerzeile benutzten. Die Menge der mit diesem Operator verwendbaren Typen wird angegeben.

Wenn Sie eine lexikalische Konstante verwenden, finden Sie weitere Hilfe unter dem Kapitel *Lexikalische Konstanten*.

1004 Sprungziel fehlt!

Sie haben die Angabe des Sprungziels vergessen.

1005 Sprungziel *xyz* nicht gefunden!

Das Sprungziel konnte in der gesamten AWL nicht gefunden werden. Überprüfen Sie, ob das Sprungziel korrekt geschrieben ist.

1006 Reserviert

1007 Konstante außerhalb des gültigen Bereichs!

Der angegebene Wert fällt nicht mehr in den Bereich des für die lexikalische Konstante ermittelten Typs. Sie können den Typ ändern, indem Sie den gewünschten Typbezeichner vor die Konstante schreiben. Dabei muss ein # den Typbezeichner von der Konstante trennen: `DWORD#60000` (vgl. Lexikalische Konstanten).

1008 Operand fehlt oder ist syntaktisch falsch!

Der Operand fehlt komplett oder er ist durch den Compiler in keiner Weise interpretierbar. Korrigieren Sie die Angabe des Operanden.

1009 ")" hat keine passende öffnende Klammer!

Jeder „)-Anweisung“ muss eine der unter Operationen mit Klammer aufgeführten Anweisungen vorausgehen.

1010 Es fehlen noch *n* schließende Klammern ")"!

Jeder Operation mit einer Klammer muss eine schließende Klammer folgen. Insgesamt wurden *n* fehlende Klammern ausfindig gemacht.

1011 Reserviert

1012 Reserviert

1013 Unbekannter Bezeichner *xyz*!

Der Bezeichner *xyz* ist dem Compiler nicht bekannt. Ein Variablenbezeichner muss vor seiner Verwendung zunächst deklariert werden. Falls er deklariert ist, ist die Schreibweise falsch.

1014 Reserviert

1015 Die Typen von Variablendefinition und Konstanten müssen identisch sein!

Für den angegebenen Wert wurde vom Compiler ein Typ ermittelt, der nicht dem Typ des hier definierten Bezeichners entspricht. Sie können den Typ ändern, indem Sie den gewünschten Typbezeichner vor die Konstante schreiben. Dabei muss ein # den Typbezeichner von der Konstante trennen: `DWORD#60000` (vgl. Lexikalische Konstanten).

1016 Der Bezeichner *xzy* ist bereits definiert!

Sie haben den Bezeichner *xyz* bereits an einer anderen Stelle definiert. Verwenden Sie einen anderen Bezeichner.

1017 Eine schreibende Operation benötigt eine Variable!

Die Operation benötigt einen beschreibbaren Variablenbezeichner als Operand.

1018 Initialisierung von Eingangsvariablen hat keinen Effekt!

Da Eingangsvariablen durch die Simulationsumgebung beschrieben werden, hat eine Initialisierung keine Wirkung. Der Initialisierungswert würde sofort wieder überschrieben (s Kapitel *Eingänge festlegen*).

1019 Die Instruktion *xyz* kann nicht mit einem Operanden aufgerufen werden!

Handelt es sich bei dieser Operation um einen unären Operator, (s. Numerische Funktionen und `NOT`), so wird er auf das aktuelle Ergebnis (*AE*, s. Verarbeitungsstapel) angewendet und benötigt daher keinen Operanden. Ansonsten handelt es sich um einen speziellen Befehl aus dem Bereich der Sprünge (JMPL, RET, …) oder dem der Typumwandlung, die naturgemäß keinen Operanden besitzen.

1020 Der Instruktion *xxx* muss mindestens eine *yyy* Instruktion folgen!

Die Anweisung *xxx* erfordert es, dass unmittelbar nach ihr mindestens eine *yyy* Anweisung folgen muss. Eine solche Anweisung ist die JMPL-Anweisung.

1021 Sprungziel *xyz* bereits definiert!

Sie haben ein Sprungziel verwendet, das bereits definiert ist. Ändern Sie den Bezeichner der Sprungmarke an den entsprechenden Stellen in der AWL.

1022 Ein-/Ausgangsvariablen (hier: *xyz*) dürfen nicht als Funktionsbausteine deklariert werden.

Eine Funktionsbausteininstanz kann nicht als eine Ein-/Ausgangsvariable dienen. Sie können nur innerhalb von `VAR` und `END_VAR` Funktionsbausteininstanzen verwenden.

1023 Eine Konstante (hier: *xyz*) kann nicht beschrieben werden!

Das Beschreiben einer Variablen, die als Konstante deklariert wurde, ist unzulässig (s. Definition von konstanten Variablen). Sie müssen, wenn wirklich diese Variable beschrieben werden soll, den Variablenbezeichner in die Variablendeklaration (s. Deklaration von Variablen) verschieben.

1024 In dem Bereich der Konstantendeklaration kann kein Funktionbaustein (hier: *xyz*) deklariert werden!

Eine Funktionsbausteininstanz kann nicht als Konstante deklariert werden. Sie können nur innerhalb von `VAR` und `END_VAR` Funktionsbausteininstanzen verwenden (s. Deklaration von Variablen).

1025 Sprungziele (hier: *xyz*) dürfen nicht innerhalb von Klammerungen stehen!

Es kann nicht in eine Klammerebene hinein gesprungen werden. Durch diese Regel ist sichergestellt, dass

der Verarbeitungsstapel konsistent bleibt.

1026 Sprunganweisungen (hier: *xyz*) dürfen nicht innerhalb von Klammerungen stehen!

Es kann nicht aus einer Klammerebene hinaus gesprungen werden. Durch diese Regel ist sichergestellt, dass der Verarbeitungsstapel konsistent bleibt.

1027 Konstanten ohne Initialisierung (hier: *xyz*) wird 0 zugewiesen!

Bei einer Konstantendefinition geht der Compiler davon aus, dass Sie der Konstanten einen Wert zuweisen wollen. Bleibt diese Zuweisung vergessen, so warnt der Compiler und Sie haben die Möglichkeit die Zuweisung vorzunehmen. Wollten Sie bewusst eine `0` zuweisen, so müssen Sie dieses ebenfalls ausschreiben, um diese Warnung zu unterbinden:

```
1   VAR CONSTANT
2       A : INT := 0;
3   END_VAR
```

1028 Eine Variable, die innerhalb VAR_INPUT definiert wurde (hier: %s), kann nicht beschrieben werden!

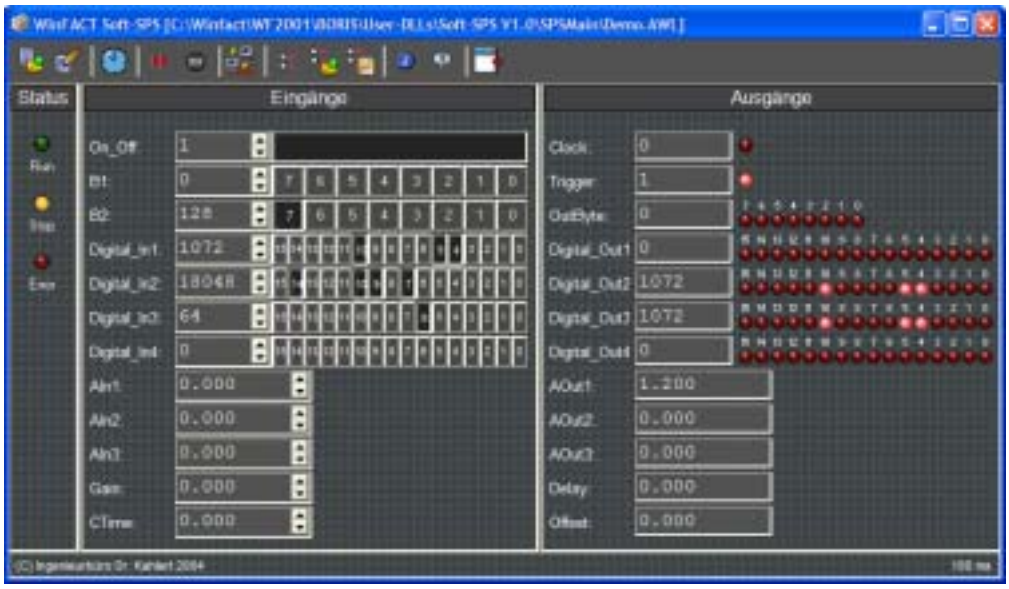
Diese Regel stellt sicher, dass die Eingangsvariablen für die gesamte AWL immer die gleiche ursprüngliche Bedeutung und den gleichen Wert besitzen. Sie müssen gegebenenfalls eine Variable deklarieren, die eine Kopie der Eingangsvariablen aufnimmt und dann mit der Kopie weiterarbeiten.

11 Arbeiten mit der Stand-Alone-SPS

Alternativ zur Nutzung innerhalb von BORIS kann die Soft-SPS auch als Stand-Alone-System benutzt werden; diese Betriebsart ist insbesondere für solche Anwender interessant, die keine BORIS-Lizenz besitzen. Die Stand-Alone-SPS wird über das Windows-Startmenü aus der WinFACT-Programmgruppe aufgerufen (siehe nachfolgende Bildschirmgrafik).

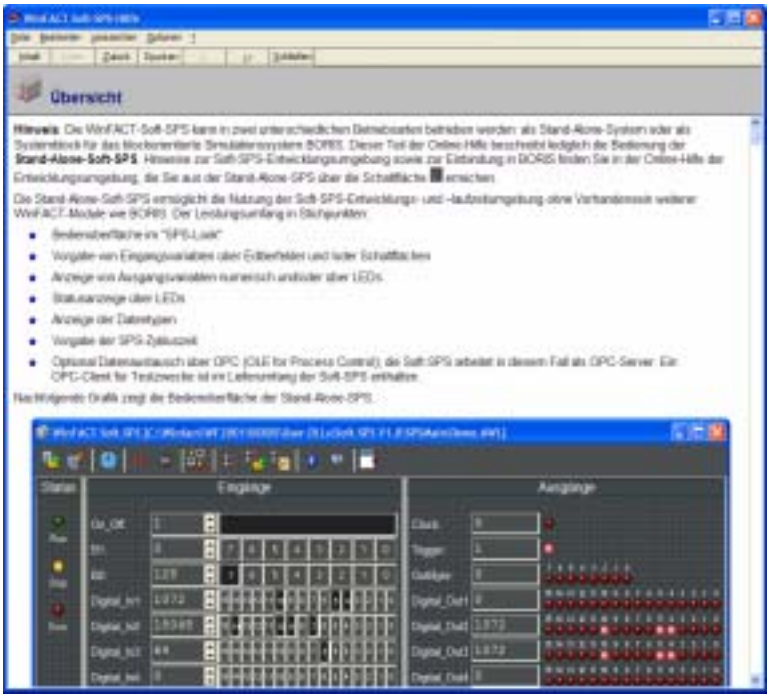
Aufruf der Stand-Alone-SPS

Nachfolgende Bildschirmgrafik zeigt die Benutzeroberfläche der Stand-Alone-SPS direkt nach dem Aufruf.



Benutzeroberfläche der Stand-Alone-SPS

Die Stand-Alone-SPS besitzt eine eigenständige Online-Hilfe, die über die Schaltfläche  aufgerufen werden kann (siehe nachfolgende Bildschirmgrafik). Hinweise zur Nutzung der Stand-Alone-SPS entnehmen Sie bitte dieser Online-Hilfe.



Online-Hilfe der Stand-Alone-SPS